Centre de recherche sur les transports (C.R.T.)
Centre for Research on Transportation

CRT-2005-30

A MULTILEVEL COOPERATIVE TABU SEARCH ALGORITHM FOR THE CAPACITATED MULTICOMMODITY NETWORK DESIGN PROBLEM

by

Ye Li

Mémoire de maîtrise

September 2005

Université de Montréal
C.P. 6128, succursale Centre-ville, Montréal QC H3C 3J7 Canada
Téléphone : (514) 343-7575 / Télécopieur : (514) 343-7121

crt@crt.umontreal.ca
www.crt.umontreal.ca

HEC MONTRÉAL · ÉCOLE POLYTECHNIQUE MONTRÉAL · Université de Montréal

105-07221/

PUBLICATION
CRT-2005-30

A MULTILEVEL COOPERATIVE TABU SEARCH ALGORITHM FOR THE CAPACITATED MULTICOMMODITY NETWORK DESIGN PROBLEM

by

Ye Li[1]

The research was supported in part by the Natural Sciences and Engineering Research Council of Canada (NSERC).

1. Department of Computer Science, University of Manitoba, 545 Machray Hall, Winnipeg, Manitoba, Canada R3T 2N2 (yeli@cs.umanitoba.ca)

Dépôt légal – Bibliothèque nationale du Québec
Bibliothèque nationale du Canada, 2005

Abstract

We describe a first multilevel cooperative cycle-based tabu search for the capacitated multicommodity network design problem. Main design challenges are associated to the specification of each search thread in cooperation, as well as to the definition of the cooperation operators. The paper proposes a first approach to address these challenges and tests it on a set of well-known benchmark problems. The proposed method appears competitive, particularly when difficult problems with high numbers of commodities are considered. Directions and challenges for future research are identified and discussed.

Key words: parallel search, multilevel cooperation, cycle-based tabu search, capacitated multicommodity network design.

Résumé

Cet article décrit un premier algorithme de recherche parallèle coopérative multiniveaux, basé sur des recherches avec tabous par des cycles, pour le problème de synthèse de réseaux multiproduits avec coûts fixes et capacités. La spécification des problèmes traités par chaque programme dans la coopération, ainsi que la définition des opérateurs de coopération comptent parmi les défis les plus sérieux associés à la conception d'un tel algorithme. Nous proposons une première approche que nous testons à l'aide d'un ensemble de problèmes présents dans la littérature. La méthode proposée offre des résultats intéressants, surtout pour les problèmes plus difficiles, avec un nombre élevé de produits. Des directions et défis de recherche sont également identifiés et discutés.

Mots clés : recherche parallèle, coopération multiniveaux, recherche avec tabou par des cycles, synthèse de réseaux multiproduits avec coûts fixes et capacités.

Acknowledgements

I would like to express my gratitude to all who helped, accompanied, supported and encouraged me to accomplish this thesis.

I want to thank the Department of Computer Science, University of Manitoba for giving me the opportunity to pursue my master's degree. I also want to acknowledge Centre de Recherche sur les Transports, Université de Montréal for allowing me to use the Centre resources and to do research work.

I am deeply indebted to my supervisors Dr. Michel Toulouse from the Department of Computer Science, University of Manitoba, and Dr. Teodor Crainic from Centre de Recherche sur les Transports, Université de Montréal. Their stimulating suggestions and guidances helped me during the research and writing of this thesis.

I am grateful to my friends, Zimpi Helen Komo and Manikandaprabhu Ganeshan, whose valuable comments, encouragements and confidence in me throughout my research supported me to complete this thesis.

Especially, I would like to give my appreciation to my parents and sister who persistently inspirate and support me not only for my research work but also in my life.

Finally, I would like to acknowledge the contribution of the Natural Sciences and Engineering Council of Canada which, through its Discovery Grant program, has funded this research project.

Contents

1 Introduction 7

2 Background 10

2.1 Definitions . . . 11

2.2 Mathematical formulation . . . 13

2.3 Neighborhood search techniques . . . 15

3 Cycle-based Tabu Search 19

3.1 Neighborhood definition . . . 19

3.2 Application of cycle-based tabu search to CMND . . . 22

3.2.1 Flow Deviation Phase . . . 24

3.2.2 Restoration Phase . . . 25

3.2.3 Intensification phase . . . 27

4 Multilevel Cooperative Search 33

4.1 Multilevel paradigm . . . 33

4.1.1 Graph partitioning problem . . . 34

4.1.2 Application of the multilevel paradigm to the graph partitioning problem . . . 36

4.2 Cooperative search . . . 42

4.3 Multilevel cooperative search . . . 44

4.3.1 Interaction operators for the graph partitioning problem . . . 44

4.3.1.1 Local partitioning 45
4.3.1.2 Local clustering 47
4.3.1.3 Interpolation 48
4.4 Summary 49

5 MCTS Algorithm for the CMND Problem **50**
5.1 Coarsening phase 51
5.1.1 Selecting arcs with fixed status 52
5.1.1.1 Fixed Cost Method 52
5.1.1.2 Frequency Method 53
5.1.2 Coarsening procedure 56
5.2 Interaction operators 59
5.2.1 Re-coarsening operator 60
5.2.2 Interpolation operator 60
5.2.3 Reverse interpolation operator 62
5.3 Modified cycle-based tabu search 63
5.4 Multilevel cooperative tabu search algorithm (MCTS) 66

6 Performance Analysis of MCTS **70**
6.1 Calibration phase 70
6.1.1 Number of levels and coarsening factors 71
6.1.2 Number of tabu iterations between activations of the interaction operators 73
6.1.3 Dimension of the elite solution set 74
6.1.4 Updating the frequency memory 75
6.1.5 Selecting fixed arcs 76
6.2 Experimentation phase 76
6.2.1 Result analysis 80
6.2.2 Search behavior analysis 83

7 Conclusions and Future Work **96**

List of Tables

6.1 Calibration of coarsening factors 73

6.2 Calibration of the number of iterations between interaction operators . . 75

6.3 Calibration of selection of initial arcs to be fixed 77

6.4 Number of iterations failed to improve the best known solution 77

6.5 Computational results 79

List of Figures

2.1 Add/Drop techniques for network design 16

3.1 Example of feasible solution 29
3.2 Example of unfeasible solution 30
3.3 Example of restoration of an unfeasible solution 31
3.4 Intensification phase 32

4.1 Cost of the 2-way partitioning after swapping vertices A and B 35
4.2 Phases of the multilevel method for the GPP 36
4.3 Coarsening phase 37
4.4 Behavior of multilevel paradigm for the graph partitioning problem (the global optimal solution is found) 40
4.5 Behavior of multilevel paradigm for the graph partitioning problem (the global optimal solution is not found) 41
4.6 Exploration of the solution space 43
4.7 Local partitioning operator 46
4.8 Local clustering operator 48

5.1 Procedure $position_FixedOpen(rn)$ 58
5.2 Procedure $position_FixedClose(rn)$ 59
5.3 Interpolation operator 62
5.4 Reverse interpolation operator 63

6.1 Number of occurrences of the best solutions at each level 86

6.2 Chart of solutions for parallel independent search for problem C45 . . . 89
6.3 Chart of solutions for MCTS and problem C45 90
6.4 Chart of solutions for MCTS and problem C56 91
6.5 Chart of solutions for MCTS and problem C61 92
6.6 Chart of solutions for MCTS and problem C45 93
6.7 Chart of solutions for MCTS and problem C56 94
6.8 Chart of solutions for MCTS and problem C61 95

List of Algorithms

1 Cycle-based Tabu Search Algorithm 23
2 Coarsening Algorithm of FC Method 55
3 Coarsening Algorithm of Frequency Method 57
4 ModifiedCycleBasedTabu ($CurrentSolutionG_i$, $EliteSetG_i$) 64
5 $Process_i$ of MCTS (network, L) 69

Chapter 1

Introduction

The Capacitated Multicommodity Network Design (CMND) problem can be represented by a weighted directed graph $G = (N, A)$ where the set of nodes N is partitioned into three non-overlapping subsets: source nodes, sink nodes and transition nodes. Each sink node represents the demand for one commodity d. The demand for commodity d is satisfied by a single source node and must transit from the source node to the sink node using arcs and transition nodes in the graph G. Each arc $(i, j) \in A$ has a capacity u_{ij}, a fixed cost f_{ij} and a routing cost c_{ij}^d. The routing cost c_{ij}^d represents the cost to transit one unit of commodity d on arc (i, j). The capacity u_{ij} is an upper bound on the number of units from different commodities that can be transitted on arc (i, j). The fixed cost f_{ij} represents the cost for having this arc in the network design. The capacitated multicommodity network design problem consists of determining $\overline{y} \subseteq A$, a subset of arcs from G which minimizes the cost to transit commodities from source to sink nodes. The subset $\overline{y}$ is a "feasible solution" if all the demands are satisfied and the total number of commodity units that transit on each arc (i, j) is less or equal to the capacity u_{ij}. The solution $\overline{y}$ is an "optimal solution" if there exists no other subset $\overline{y}' \subseteq A$ for which the

sum of the fixed costs and routing costs is smaller than the one for the subset $\overline{y}$.

The capacitated multicommodity network design problem plays a significant role in modeling many network related applications. For example, it is used to design telephone networks, railway networks or computer networks in order to minimize the cost of communications, transportation of merchandises or the cost to upload and download data, images and videos. The CMND is a core problem in many areas of Operations Research, Computer Science, Applied Mathematics, Engineering and Management. Unfortunately, there is no known efficient approach to solve the CMND problem using computers. In fact, the CMND problem has been proved to be NP-hard [1, 24, 25], therefore it is unlikely that we will ever be able to solve it exactly for large problem instances.

As usually the case for NP-hard problems, in order to solve them, one has to settle for a feasible solution which is not necessary optimal. Different approaches for the CMND problem have been proposed in the literature: simplex-based cutting plane methods [3], Lagrangian relaxations [5], and neighborhood search heuristics [7, 12]. In this thesis, we propose an approach called Multilevel Cooperative Tabu Search (MCTS) which combines neighborhood search and parallel computing to obtain better approximations of optimal solutions for the CMND problem. More specifically, we have adapted the best known neighborhood search heuristic for the CMND problem [12] to be used as the search subroutine of a multilevel cooperative algorithm for the CMND problem. The main contributions of this thesis are the followings:

1. the adaptation of the cycle-based tabu search to multilevel search;

2. the definition and implementation of a strategy providing an initial set of search spaces of a CMND problem instance to be searched in parallel by independent cycle-based tabu search processes;

3. the design of operators which refine search spaces and reinitialize the exploration in each search space by a cycle-based tabu search procedure.

The thesis is organized as follows: Chapter 2 provides an intuitive description of the CMND problem followed by its mathematical formulation as well as general introduction to the application of neighborhood search methods to this problem. Chapter 3 describes the cycle-based tabu search algorithm. Chapter 4 introduces the multilevel cooperative method through an illustration of the application of this method to the hypergraph partitioning problem. Chapter 5 describes the multilevel cooperative algorithm for the CMND problem. Parameter calibrations are described in Chapter 6, which also reports and analyzes the results of our experimentations. The conclusion and potential research directions are provided in Chapter 7.

Chapter 2

Background

The capacitated multicommodity network design problem is a mixed-integer programming problem, with real and integer decision variables. The integer (0-1) decision variables represent the choice of including or not in the design a specific arc (i, j) from the graph G. The continuous decision variables represent an network flow optimization problem that needs to be solved for each network design. The CMND problem is a combinatorial optimization problem (in its design arcs) with an underlying network flow problem. These two main optimization structures are keys to understanding the mathematical formulation of the problem as well as in the solution strategies that are proposed to solve it. In this chapter, we first define the underlying problems associated with CMND. Next, we introduce the mathematical formulation of the problem that will be used throughout this thesis. Finally, we provide background on applications of neighborhood search methods to solve the CMND problem.

2.1 Definitions

Consider the following scenario: "suppose wheat is stored in warehouses in Winnipeg and Saskatoon which needs to be transported to customers in Montreal, Ottawa and Halifax. Each warehouse can ship up to 1500 units. Montreal, Ottawa and Halifax have customers which require 850, 700 and 600 units, respectively. Wheat would have to pass through other cities such as Thunderbay or Chicago before getting to their various destinations."

Applications involving transportation of goods from sources to destinations can be described using a mathematical model called the *network flow problem*. The network flow problem can be described using a directed graph. Arcs in the graph stand for the routes in the application that can be used to transport commodities such as wheat. Generally speaking, nodes in the graph are partitioned in three categories: *source* nodes, which indicate the origin points (cities such as Winnipeg and Saskatoon as shown in the above example); *sink* nodes, which indicate the destination points (cities such as Montreal, Ottawa and Halifax); and finally, *transition* nodes, which represent points (the cities of Thunderbay and Chicago) through which the routes have to pass.

Transporting wheat from its origin to the required destination incurs a cost, called a *routing cost*, which represents the cost per unit of commodity transported. In the model, we associate to each arc the cost of routing one unit of flow of the commodity. The network flow problem models some basic resource and associated costs which can then be used to express the overall cost of a specific transport application. The optimization version of network flow problem, i.e., finding routes that minimize the overall cost of a transportation application, is the *minimum cost flow problem*.

Wheat has to be conveyed using some transportation equipments (trucks, ships, trains, or airplanes) and each equipment has a limited carrying capacity. Hence, in the

corresponding model, each arc also has an associated capacity that denotes the maximum amount of flow traveling on the arc. This problem can be termed as the *capacitated minimum cost flow problem*.

Instead of transporting a single commodity such as wheat, one may envision a network flow problem where several kinds of commodities have to be transported simultaneously using the same underlying network. The problem of transporting more than one commodity at minimum cost is the *capacitated multicommodity minimum cost flow (CMCF) problem*. In the CMCF problem, each commodity has its own origins and destinations. Furthermore, to each arc and to each commodity is associated a different cost per unit of flow. Hence, it increases the difficulty of the problem. The objective of CMCF problem is to allocate the limited capacity of each arc to the different commodities in order to minimize the cost of transporting the goods from origins to destinations. The CMCF problem is a linear programming problem that can be solved using a commercial software called CPLEX. It is routinely used to solve this problem computationally. The CMCF problem is represented by the continuous decision variables in the CMND problem. This sub-problem needs to be solved for each network design.

In some transportation applications, one needs to take into account the cost of building/renting the transportation infrastructure associated to each arc in the network flow graph. The cost associated to the transportation infrastructure is termed the *fixed cost*. Hence, the whole system cost which aims to be minimal, must include the routing cost and the fixed cost. This type of transportation application is modeled as the *capacitated multicommodity network design* (CMND) problem and it consists of selecting a subset of arcs in the network that minimizes the whole system cost (the sum of the fixed cost of building system infrastructure and cost of routing commodities), while satisfying the transportation requirements for several commodities simultaneously.

2.2 Mathematical formulation

Let $G = (N, A)$ be a directed graph where N is a set of nodes and A a set of directed arcs. Without loss of generality, we assume that all arcs $(i, j) \in A$ are *design arcs*, i.e., arcs that can be opened (included in the network solution) or closed (excluded from the network solution) in order to form a design (solution) of the problem. We provide an arc-based mathematical formulation of CMND problem:

$$\min \xi(x, y) = \sum_{d \in D} \sum_{(i,j) \in A} c_{ij}^d x_{ij}^d + \sum_{(i,j) \in A} f_{ij} y_{ij}$$

$$\text{where} \quad \sum_{j \in N^+(i)} x_{ij}^d - \sum_{j \in N^-(i)} x_{ji}^d = \begin{cases} r^d & \text{if } i = o(d) \\ -r^d & \text{if } i = s(d) \\ 0 & \text{otherwise} \end{cases} \quad \forall i \in N, \forall d \in D \qquad (1)$$

$$\sum_{d \in D} x_{ij}^d \leq u_{ij} y_{ij} \qquad \forall (i, j) \in A \qquad (2)$$

$$x_{ij}^d \geq 0 \qquad \forall (i, j) \in A, \forall d \in D \qquad (3)$$

$$y_{ij} \in \{0, 1\} \qquad \forall (i, j) \in A \qquad (4)$$

The meaning of the variables is the following:

- D : the set of commodities to be moved over network G.
- d : one commodity $d \in D$.
- $o(d)$: each commodity d has a single origin $o(d)$.
- $s(d)$: each commodity d has a single destination $s(d)$.
- r^d: for each commodity d, there is a flow demand of r^d units

y_{ij}: the design variable that is equal to 1 if arc (i, j) is chosen in the design and equal to 0 if it is not chosen.

x_{ij}^d: the flow distribution decision variable that denotes the amount of flow of commodity $d \in D$ on arc (i, j).

f_{ij}: the fixed cost of arc (i, j).

c_{ij}^d: the routing cost of one unit flow of commodity d on arc (i, j).

u_{ij}: capacity of arc (i, j).

$N^+(i)$: the set of nodes j such that (i, j) is the arc from node i to j.

$N^-(i)$: the set of nodes j such that (j, i) is the arc from node j to i.

The objective function $\xi(x, y)$ which must be minimized, represents the system design cost that includes the fixed cost of the arcs that are chosen in a design and the routing cost which is proportional to the number of units of commodities. Constraint (1) shows the network flow conservation. That is, for each source node i, there is only the flow of r^d units for commodity d leaving this node i; for each sink node i, there is only the flow of r^d units for commodity d entering the node i; for each transition node i (other than source nodes or sink nodes) and commodity d, the amount of the flow entering node i is equal to the amount of the flow leaving node i. Constraint (2) denotes, for each arc (i, j), the total flow of all the commodities can not exceed its capacity. Constraint (3) illustrates that the flow distribution decision variable x_{ij}^d is not negative while constraint (4) shows that the design variable y_{ij} is either 0 or 1.

From this mathematical formulation, we notice that CMND is a capacitated multicommodity minimum cost flow problem (CMCF) once a design $\overline{y}$ is obtained. This CMCF can be written as

$$\min \xi(x(\overline{y})) = \sum_{d \in D} \sum_{(i,j) \in A(\overline{y})} c_{ij}^d x_{ij}^d$$

$$\text{where} \quad \sum_{j \in N^+(i)} x_{ij}^d - \sum_{j \in N^-(i)} x_{ji}^d = \begin{cases} r^d & \text{if } i = o(d) \\ -r^d & \text{if } i = s(d) \\ 0 & \text{otherwise} \end{cases} \quad \forall i \in N, \forall d \in D$$

$$\sum_{d \in D} x_{ij}^d \leq u_{ij} y_{ij} \qquad \forall (i,j) \in A(\overline{y})$$

$$x_{ij}^d \geq 0 \qquad \forall (i,j) \in A(\overline{y}), \forall d \in D$$

where $A(\overline{y})$ symbolizes the set of arcs corresponding to the design $\overline{y}$. Therefore, we said that CMND is the combinatorial problem of choosing of subgraph over the original network plus the optimization problem of the CMCF problem. That is, a solution of CMND problem can be seen as a design $\overline{y}$, assigning a value of 0 or 1 to each design variable, plus the optimal flow $x^*(\overline{y})$ of the corresponding CMCF problem. Hence, the objective function $\xi(x, y)$ can be also written as:

$$\xi(\overline{y}, x^*(\overline{y})) = \sum_{(i,j) \in A(\overline{y})} f_{ij} \overline{y}_{ij} + \xi(x^*(\overline{y}))$$

Even though considering only the fixed cost, the CMND is NP-hard because it has to consider all the possible subgraphs that satisfy the constraints of the transportation. In the CMND problem, this combinatorial problem of the subgraphs is combined with the CMCF problem. This makes the CMND problem even more difficult to solve. No exact method exists that can solve the CMND problem in a reasonable amount of time for application instances of a realistic size.

2.3 Neighborhood search techniques

The first search heuristics proposed for the capacitated network design problem were simple add/drop procedures. These procedures evaluate the impact on the cost function

of adding or removing an arc from the design [21, 27]. Given a current feasible solution, add/drop heuristics compute the cost of all feasible solutions that can be obtained by adding or removing a single arc from the current solution. The set of feasible solutions that can obtained in this way form the *neighborhood* of the current solution. In Figure 2.1, solid arcs represent a current feasible solution of an instance of the network design problem while dotted lines represent the arcs that are not chosen in the design. Two solutions in the neighborhood of the current solution can be obtained by adding arc (C, F) or dropping arc (C, S). Once the cost of all neighbors has been computed, add/drop procedures select the solution with the lowest cost (best improvement). If the cost of the chosen solution is better than the current solution, the chosen solution becomes the new current one and the neighborhood evaluation is repeated. Eventually, add/drop heuristics reach a point where there is no improving solution in the neighborhood of the current solution. Then, the add/drop heuristic stops and returns the current solution as the best solution to a network design problem instance.

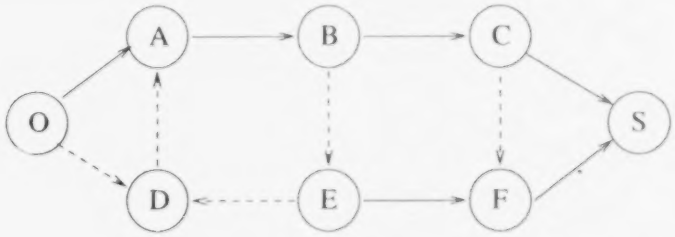


Figure 2.1: Add/Drop techniques for network design

Add/drop procedures only make moves which improve the cost function. This is a serious drawback of this method, limiting their capacity to explore the solution space. For example, once an add/drop procedure reaches the first local optimum, the search will

end since, by the definition of a local optimum, no add or drop move can improve the cost function. To address this kind of limitations, Crainic, Gendreau and Farvolden [7] have applied the tabu search metaheuristic to the CMND problem. Tabu search is a neighborhood search heuristic which, unlike pure add/drop procedures, can explore beyond local optima by making moves which do not necessarily improve the cost of the current solution. During the period where the cost function is degraded, tabu search seeks to find a new plateau (or valley) in the solution space from which the sequence of improving moves can resume. This leads to a more extensive exploration of the solution space, yielding better solutions.

Neighborhoods based on adding (dropping) arcs are not ideal for a problem like CMND. For example, in Figure 2.1, adding arc (E, D) into the network will not produce any significant change to the cost function since adding arc (E, D) does not generate any new path between the origin and the destination (there is only one path $\{(O, A), (A, B), (B, C), (C, S)\}$ between the origin (node O) and the destination (node S). This situation will not be changed by adding arc (E, D)). In this context, Ghamlouche, Crainic and Gendreau [12] have proposed a new tabu search heuristic for the CMND problem with a completely different type of neighborhood, called cycle-based neighborhood. Cycle-based neighborhood is a key component of proposed algorithm. Details about this neighborhood structure will be discussed in Chapter 3.

Another heuristic for the CMND problem has been proposed recently which combines the cycle-based tabu search and the path relinking approach [13]. The cycle-based tabu search produces promising solutions, called *Elite solutions*, which are stored in a set called the *reference set*. The aim of the path relinking procedure is to find a path between two elite solutions in the reference set in order to intensify the exploration in the solution space of elite solutions. The solutions in the path are obtained by introducing attributes of elite solutions in the reference set to guide the search. Experimental results

indicate that the proposed path relinking method currently offers the best performance for the CMND problem.

Finally, the computation of neighborhood search heuristics can be speed up in different ways through parallel implementations of the heuristics [8]. One parallelization technique consists of the initiation of several concurrent tabu search processes from different regions of the solution space. During the computation, processes exchange information about their own exploration of the solution space. These parallelization techniques are called *cooperative searches*, and they have been applied in parallelization of the tabu search method [34]. A parallel cooperative tabu search method has been proposed and tested recently for the CMND problem [6]. As shown in [35], search parameters and cooperation interact with each other in a very complex manner. This makes it difficult to predict the search behavior of the individual processes. For example, it is difficult to prevent the search paths from overlapping with one another. In this thesis, I will use a new approach for cooperation, *parallel multilevel cooperative tabu search*, which provides a greater control on how search parameters and cooperation impact on each other.

Chapter 3

Cycle-based Tabu Search

In this chapter, we describe the cycle-based tabu search algorithm for the CMND problem [12]. A variation of this algorithm will be used as a search heuristic in our multilevel cooperative algorithm for the CMND problem. The key feature of the cycle-based tabu search algorithm is its neighborhood structure, called the cycle-based neighborhood. We first describe this neighborhood structure. Then, we elaborate on the tabu search algorithm with the cycle-based neighborhood structure for the CMND problem.

3.1 Neighborhood definition

Let $G = (N, A)$ be an directed graph representing a network design. A *path* in G is a sequence of closed or opened arcs such that the origin node of each arc is the destination node of the preceding arc in the sequence and no vertex appears more than once. That is, all arcs on a path follow the same direction. For example, in Figure 3.1 (a), the sequence $\langle (A, B), (B, C), (C, G) \rangle$ is a path. We define a *chain* like a path where the arcs do not necessary follow a single direction. A *cycle* is a closed chain. In Figure 3.1 (a), the

chain $\langle (E,B),(B,C),(C,F),(E,F)\rangle$ form a cycle.

A cycle in a cycle-based neighborhood structure is composed of two paths connecting the same pair of nodes. For example, cycle $\langle (E,B),(B,C),(C,F),(E,F)\rangle$ in Figure 3.1 (a) contains two paths $\langle (E,B),(B,C),(C,F)\rangle$ and $\langle (E,F)\rangle$ which connect to the same pair of nodes E and F. The purpose of the two paths forming a cycle is to allow for the flow of commodities on one path of the cycle to be deviated on the other path of this same cycle. Deviating flow in such cycles change the cost of the current solution $\overline{y}$, and provide a neighbor to $\overline{y}$. However, the number of neighbors based on flow deviation can be huge, which makes the computation time to evaluate the neighborhood excessive. To reduce the cost of computing neighbors, only those flow deviations that close at least one opened arc in the current solution are considered.

Let

$$\Gamma(\overline{y}) = \{\sum_{d\in D} x_{ij}^d > 0 : (i,j) \in A(\overline{y})\}.$$

be the set of the different volumes of flow on the open arcs in the current solution $A(\overline{y})$. The set $\Gamma(\overline{y})$ gives us the size of the flow deviations that can constitute neighbor solutions. Let the *residual capacity* of a cycle to be the maximum flow which can be deviated around a cycle. To deviate a flow of value γ on a given cycle, the residual capacity of this cycle should be at least γ. For each value $\gamma \in \Gamma(\overline{y})$, we compute the γ-*residual network*, which is a network that identifies all the cycles capable of deviating at least γ units of flow. The γ-residual network is obtained from the network corresponding to the current solution $\overline{y}$ by replacing each arc in the network $\overline{y}$ by at most two arcs $(i,j)^+$ and $(j,i)^-$. Arc $(i,j)^+$ represents γ units of flow that can be added to arc (i,j). Arc $(j,i)^-$ represents γ units of flow that can be eliminated from arc (i,j). An arc $(i,j)^+$ is included in the γ-residual network iff the residual capacity (i.e., the subtraction of the capacity and the flow amount) of arc (i,j) is greater or equal to γ. Arc $(j,i)^-$ is included in the γ-residual network iff the total flow on arc (i,j) is greater or equal to

γ, i.e. $\sum_{d \in D} x_{ij}^d \geq \gamma$. Figure 3.1 (b) shows an example of 2-residual network for the current network design $\overline{y}$ shown in Figure 3.1 (a).

Among all the cycles in a γ-residual network, there exists at least one arc $(i,j)^-$ corresponding to an opened arc (i,j) in the network $\overline{y}$ with total flow γ. The arc (i,j) can be closed, and its flow can be deviated in the network $\overline{y}$ on the alternate path of one of the cycles in which $(i,j)^+$ is involved in the γ-residual network.

From a γ-residual network, we can extract all the neighbors of the current solution $A(\overline{y})$ that can be obtained by deviating γ units of flow. Then, the cost of each of these neighbors is computed as follows: For each arc $(i,j)^+ \in \gamma$-residual network, we find an approximation c_{ij}^+ of the increase in cost from routing γ units of additional flow on this arc:

$$c_{ij}^+ = \begin{cases} \frac{\sum_{p \in P} c_{ij}^p}{|P|}\gamma + f_{ij} & \text{if } \sum_{p \in P} x_{ij}^p = 0; \\ \frac{\sum_{p \in P} c_{ij}^p}{|P|}\gamma & \text{if } \sum_{p \in P} x_{ij}^p > 0; \end{cases}$$

Similarly, for each arc $(j,i)^- \in \gamma$-residual network, we compute an approximation c_{ij}^- of the decrease in cost from the reduction of γ units of flow on this arc:

$$c_{ji}^- = \begin{cases} -\frac{\sum_{p \in P} c_{ij}^p x_{ij}^p}{\sum_{p \in P} x_{ij}^p}\gamma - f_{ij} & \text{if } \sum_{p \in P} x_{ij}^p = \gamma; \\ -\frac{\sum_{p \in P} c_{ij}^p x_{ij}^p}{\sum_{p \in P} x_{ij}^p}\gamma & \text{if } \sum_{p \in P} x_{ij}^p > \gamma; \end{cases}$$

The cost of a neighbor solution is computed by adding the cost of a flow deviation to the cost of the current solution. Figure 3.1 (c) is a 2-residual network with the changes on costs after deviating γ units of flow on each arc on the network $\overline{y}$ shown in Figure 3.1 (a).

Next, the move in the neighborhood is computed by identifying the cycle that maximize the cost reduction or minimize the cost increase. Let $C(\gamma)$ be the set of arcs in the current solution $\overline{y}$ that can support the movement of γ units of flow. That means, for

open arcs, the flow in the current solution is at least γ, and for closed arcs, the residual capacity is at least γ. In Figure 3.1 (d), for $\gamma = 4$, if we assume that C is a set of all the arcs in the network, $C(4)$ includes arcs (E, D), (D, A), (A, B) and (B, C). The procedure to identify the best neighbor is the following one:

1. For each $\gamma \in \Gamma(\overline{y})$
   (a) For each arc $(i, j) \in C(\gamma)$
      i. Find the cycle with the lowest cost that include arc (i, j).

This procedure considers each arc $(i, j) \in C(\gamma)$ as the beginning of a cycle. For each arc (i, j), the procedure looks at all the cycles that contain arc (i, j) and identifies the cycle with the lowest cost.

3.2 Application of cycle-based tabu search to CMND

The cycle-based tabu algorithm that will be used in this thesis is illustrated in Algorithm 4. First, an initial solution is generated by opening all the arcs and solving the corresponding capacitated multicommodity minimum cost flow (CMCF) problem. This initial solution is then assigned to be the $CurrentSolution$ and $BestSolution$.

The tabu search procedure with cycle-based neighborhood is implemented through a while loop with a stopping criterion that is either a pre-defined number of iterations or a time limit. At each iteration in the loop, there are flow deviation, restoration and intensification phases.

Algorithm 1 Cycle-based Tabu Search Algorithm

// Initialization

Generate the initial solution and assign it to $BestSolution$ and $CurrentSolution$;

while (a stopping criterion is not satisfied)

1. Determine $\Gamma(\overline{y})$ and C for the current solution $\overline{y}$;
2. **for** each $\gamma \in \Gamma(\overline{y})$

   Install the γ_residual network;

   Determine $C(\gamma)$;

   for each arc $(i, j) \in C(\gamma)$

   Find the associated "low cost" cycle by using the network optimization procedure;

   Find the lowest cost cycle over all $(i, j) \in C(\gamma), \gamma \in \Gamma(\overline{y})$;
3. Move to the new solution by opening or closing corresponding arcs;
4. Evaluate the solution value of the new solution by solving the associated CMCF problem using CPLEX;
5. Trim, i.e., close opened arcs with no flow;
6. Set the tabu status to each complemented arc;
7. **if** the solution is not feasible

   Execute a $restoration$ phase;
8. Perform an $intensification$ phase if current solution satisfies:

   $\frac{CurrentSolution - BestSolution}{BestSolution} \leq IntensGap$;
9. **if** $currentSolution < BestSolution$

   Update $BestSolution$;

3.2.1 Flow Deviation Phase

At the beginning of each iteration, the set of flow deviation values $\Gamma(\overline{y})$ and the set of candidate arcs C are given. In this thesis, C is always a set of closed arcs. Next, for each value $\gamma \in \Gamma(\overline{y})$, a γ_residual network is constructed and $C(\gamma) \in C$ is determined over the γ_residual network. For each arc (i, j), an associated "lowest cost" cycle is found in the γ_residual network. The lowest cycle amongst all the cycles for all arcs $(i, j) \in C(\gamma)$ and $\gamma \in \Gamma(\overline{y})$ is set as the best move to the next solution (new design) by opening and closing the appropriate arcs over the best cycle. The cost of the new solution is computed by solving the CMCF problem using CPLEX. This new solution may have opened arcs without flow. Hence, the design is then adjusted by closing these opened arcs. This phase of adjustment, called trim, reduces the cost of the solution by eliminating the fixed cost of arcs without flow. A short-term tabu memory is then used to prevent the search from cycling by recording tabu status for the arcs that are opened or closed during each iteration.

For instance, Figure 3.1 (a) shows a design with $(D, G, 2)$ and $(E, F, 2)$ which represent source nodes, sink nodes and demands of commodities d_1 and d_2, respectively. The current solution routes 2 units of flows of commodity d_1 on path $\langle (D, B), (B, C), (C, G) \rangle$ and 2 units of flows of commodity d_2 on path $\langle (E, B), (B, C), (C, F) \rangle$. The value of the objective function of the current solution is 61. Let $\Gamma(\overline{y}) = \{2, 4\}$ and $C = \{(A, B), (D, A), (E, D), (E, F), (B, F)\}$. Thus, $C(2) = C = \{(A, B), (D, A), (E, D), (E, F), (B, F)\}$ on 2_residual network (shown in Figure 3.1 (c)), and $C(4)$ $= \{(E, D), (D, A), (A, B)\}$ on 4_residual network (shown in Figure 3.1 (d)). In the 2_residual network (shown in Figure 3.1 (c)), in $C(2)$, cycles starting with candidate links (D, A) are $\langle (A, B), (B, D), (D, A) \rangle$ and $\langle (A, B), (B, E), (E, D), (D, A) \rangle$ and costs are 3 and -3 respectively; cycle starting with candidate link (E, F) is $\langle (E, F)$

(F,C), (C,B), $(B,E)\rangle$ which has cost 11; cycle starting with candidate link (B,F) is $\langle(F,C)$, (C,B), $(B,F)\rangle$ with cost 2. There is no cycle starting with candidate links in $C(4)$ by observing Figure 3.1 (d). The lowest cost cycle is $\langle(A,B)$, (B,E), (E,D), $(D,A)\rangle$ found on 2_residual network in Figure 3.1 (b) with a cost improvement of -3. This best move results in the new network design (shown in Figure 3.1 (e)) by opening arcs (E,D), (D,A) and (A,B) and closing arcs (E,B) on the current network design (shown in Figure 3.2 (a)).

3.2.2 Restoration Phase

Flow deviations on the best cycle may yield a solution that is not feasible since there may be no paths between source nodes and sink nodes of some commodities. For example, Figure 3.2 (a) shows a design with $(D,F,2)$, $(C,G,2)$ and $(A,B,2)$ which represent source nodes, sink nodes and demands of commodities d_1, d_2 and d_3, respectively. The current solution routes 2 units of flows of commodity d_1 on path $\langle(D,E)$, $(E,F)\rangle$, 2 units of flows of commodity d_2 on path $\langle(C,E),(E,F),(F,G)\rangle$ and 2 units of flows of commodity d_3 on path $\langle(A,D),(D,E),(E,B)\rangle$. The value of the objective function of the current solution is 55. Let $\Gamma(\overline{y}) = \{2,4\}$ and $C = \{(D,H)$, $(H,F)\}$. We assume $C(2) = C = \{(D,H)$, $(H,F)\}$ on 2_residual network (shown in Figure 3.2 (b)) and $C(4) = C = \{(D,H)$, $(H,F)\}$ on 4_residual network (shown in Figure 3.2 (c)). In the 2_residual network (shown in Figure 3.2 (b)), cycles starting candidate links in $C(2)$ are $\langle(D,H)$, (H,F), (F,E), $(E,D)\rangle$ with cost 4. The cycles starting with candidate links in $C(4)$ are $\langle(D,H)$, (H,F), (F,E), $(E,D)\rangle$ with cost -5 (shown in Figure 3.2 (c)). The lowest cost cycle is $\langle(D,H)$, (H,F), (F,E), $(E,D)\rangle$ found on 4_residual network in Figure 3.2 (c) with a cost improvement of -5. This best move results in the new network design (shown in Figure 3.2 (d)) by opening arcs (D,H) and (H,F) and

closing arcs (D, E) and (E, F) on the current network design (shown in Figure 3.2 (a)). This network design is unfeasible since there are no paths between the source node and sink node for commodities d_2 and d_3. To handle unfeasible solutions, this algorithm uses the artificial arcs between source nodes and sink nodes for each commodity. In the example above, the flow of commodities d_2 and d_3 will be deviated on those artificial arcs which have a capacity value equal to the demand of the associated commodity and very high routing cost. High routing cost is set to make sure that the artificial arcs are undertaken only if there are no feasible solutions. So, when the solution cost is very high, it is likely that the solution includes artificial arcs, i.e., this solution is unfeasible.

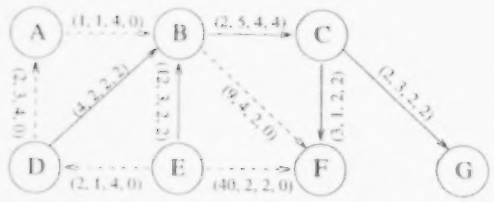
When a solution is unfeasible, a *restoration phase* is used to bring the search back to the feasible regions of the solution space. It is achieved by deviating the flows from artificial arcs to network arcs. Restoration operates similarly to steps 1 to 6 (in Algorithm 4) except that the set C is restricted to the artificial arcs with flows, while the set γ is restricted to the flow of commodities of these artificial arcs. Figure 3.3 describes how the restoration phase works. Figure 3.3 is based on the same problem as in Figure 3.2 wich routes respectively 2 units of flows for commodity d_1 (from D to F), d_2 (from C to G) and d_3 (from A to B). In Figure 3.3 (a), the infeasible solution has two artificial arcs with zero fixed cost, high routing cost (costs higher than 400000 are defined as "high"), capacity 2 and flow 2. In Figure 3.3 (a), an unfeasible solution is found due to its "high" cost 1600010 when it routes 2 units of flows for commodity d_1 on path $\langle (D, H), (H, F) \rangle$, routes 2 units of flows for commodity d_2 on artificial arc (C, G) and routes 2 units of flows for commodity d_3 on artificial arc (A, B). In this example, $\Gamma(\overline{y}) = \{2\}$ and $C = C(2) = \{(A, B), (C, G)\}$. Hence, in the 2-residual network shown in Figure 3.3 (b), artificial arcs (A, B) and (C, G) are replaced only by $(j, i)^-$ since the capacity and the flow amount of every artificial arc are equal to γ. Hence, no matter which cycle is chosen to be the best move, artificial arcs are closed af-

ter the best move. Subsequently, flows are deviated from artificial arcs to another set of non-artificial arcs. For example, Figure 3.3 (b) shows cycle starting with candidate link (A, B) is $\langle (B, A), (A, D), (D, E), (E, B) \rangle$ with cost -799977 and cycle starting with candidate link (C, G) is $\langle (C, E), (E, F), (F, G), (G, C) \rangle$ with cost -799974. Hence, the cycle $\langle (B, A), (A, D), (D, E), (E, B) \rangle$ is chosen to be the best move and it generates the new network design shown in Figure 3.3 (c) when opening arcs (A, D), (D, E) and (E, B), and closing arc (A, B). In this new network design, artificial arc (A, B) is removed. But, we know this solution is still infeasible since the artificial arc (C, G) exists in this solution. Hence, the restoration phase is executed again and 2_residual network is built as shown in Figure 3.3 (d). The only cycle is $\langle (C, E), (E, F), (F, G), (C, G) \rangle$ since $C = C(2) = \{(C, G)\}$. Finally, Figure 3.3 (e) displays in the network design, the artificial arc (C, G) is removed as well.

3.2.3 Intensification phase

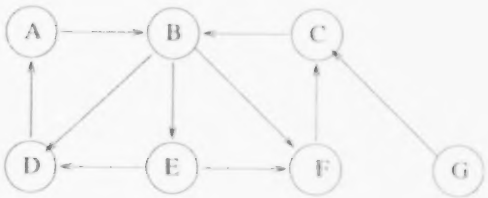
In the deviation and restoration phases, the flow of all the commodities is deviated on the new best cycle. When the deviation phase or restoration phase finds a "good" solution that either improves the best solution or is close to it by a pre-defined percentage (called $IntensGap$), usually this solution may be improved further by iteratively modifying the flow distribution of one commodity at a time. The flow of one commodity can be modified using the same flow deviation approach described in Section 3.2.1. This flow deviation of individual commodities is called the *intensification phase* of the cycle-based tabu search algorithm as it refines solutions obtained in the flow deviation and restoration phases. The flow deviation for each commodity is obtained by generating the γ-residual networks for this commodity and proceeding exactly as for the flow deviation and restoration phases. The only difference is that during intensification phase, only

negative cycles are considered. The pseudo-code of the intensification phase is shown in Figure 3.4. Intensification stops once there are no more negative-cost cycles. Moves around negative-cost cycles ensure that the feasible solution is always better than the current one.

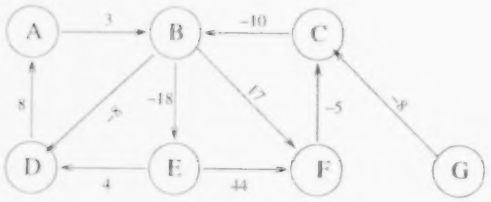


(FixedCost, RoutingCost, Capacity, Flow)

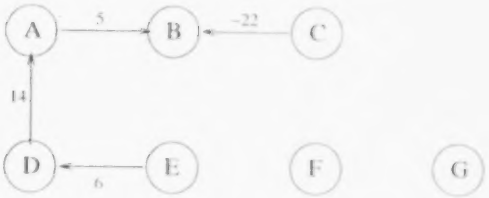
(a) A Network Design



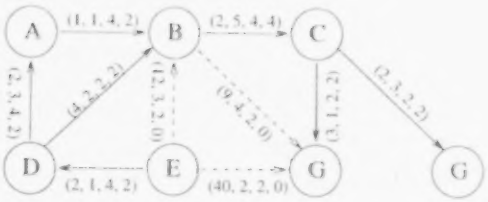
(b) 2_residual Network without Costs



(c) 2_residual Network with Costs



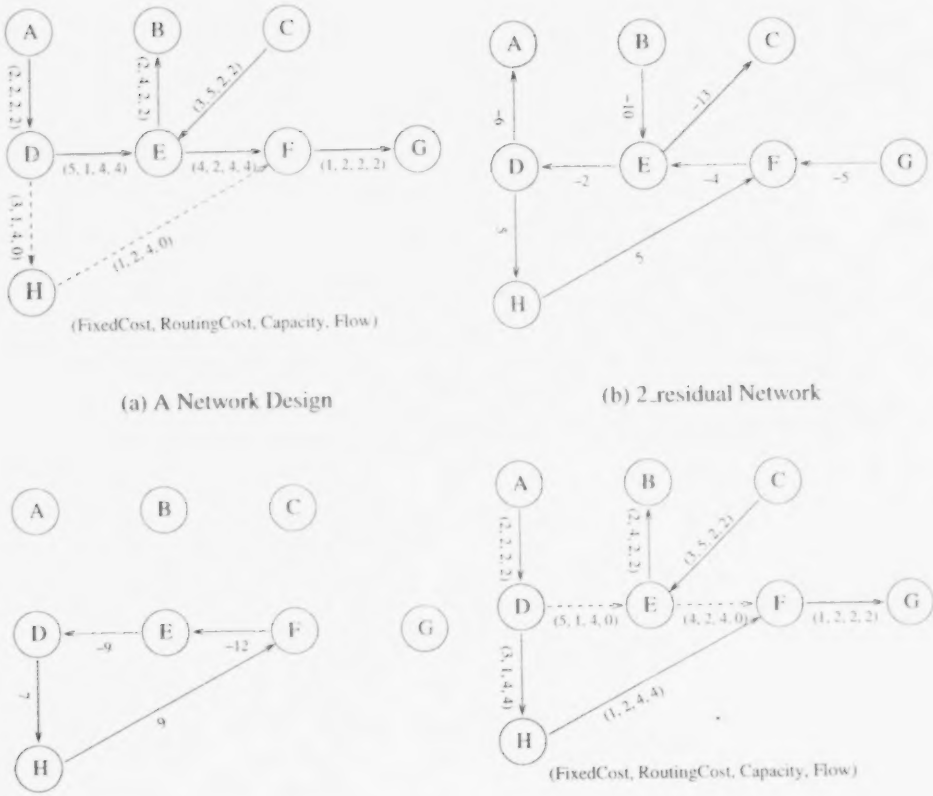
(d) 4_residual Network



(FixedCost, RoutingCost, Capacity, Flow)

(e) The New Network Design After The Best Move

Figure 3.1: Example of feasible solution



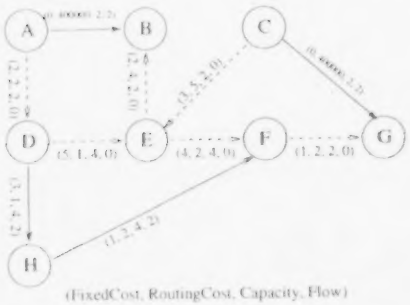
(a) A Network Design

(b) 2_residual Network

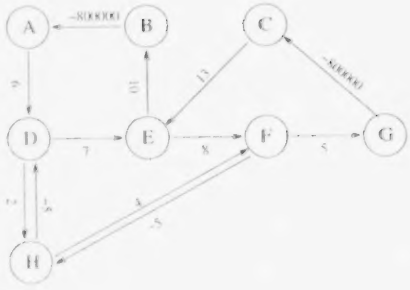
(c) 4_residual Network

(d) The New Network Design After The Best Move

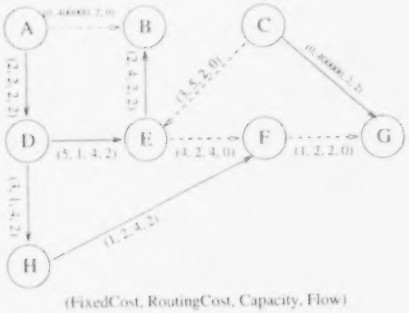
Figure 3.2: Example of unfeasible solution



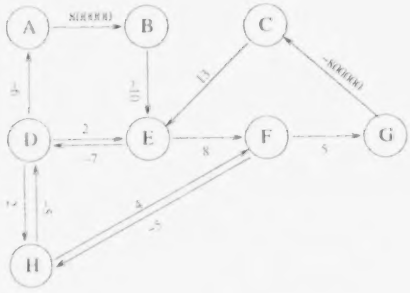
(a) A Network Design Which is Unfeasible



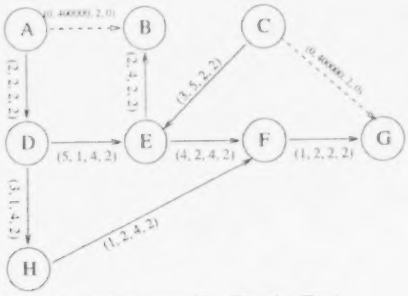
(b) 2_residual Network



(c) The New Network After The Best Move



(d) 2_residual Network of The New Network



(e) Another New Network

Figure 3.3: Example of restoration of an unfeasible solution

while (no negative cost cycle is detected in any γ_residual network)
  for (each commodity $d \in D$ and $\gamma \in \Gamma^d$)
    install the γ_residual network;
    determine $C(\gamma)$;
    for each arc $(i, j) \in C(\gamma)$ and $\gamma \in \Gamma^d$
      find the low cost cycle by using the network optimization procedure;
    find the lowest cost cycle over all $(i, j) \in C(\gamma), \gamma \in \Gamma^d$;
    γ_{best} denotes the associated volume of deviated flow;
    if the lowest cost cycle is negative cost (improving)
      modify the flow around the best cycle by γ_{best};
      move to the new solution by opening or closing corresponding arcs;
      update the cost of current solution by adding the cost of the cycle
      (without solving CMCF problem);
      set the tabu status to each complemented arc;
evaluate the current solution value by solving the associated CMCF problem;

Figure 3.4: Intensification phase

Chapter 4

Multilevel Cooperative Search

Multilevel cooperative search is a new parallel meta-heuristic developed by Toulouse et al [26, 36]. This meta-heuristic combines some aspects of the multilevel design paradigm for discrete optimization problems first introduced by [2, 14] and cooperative search algorithms [34]. In this chapter, we give a brief overview of the multilevel and cooperative search paradigms. Then, we describe in more details the multilevel cooperative search approach. The multilevel paradigm will be illustrated using the graph partitioning problem (GPP). Therefore, we first formally define this problem and introduce neighborhood search techniques for this problem.

4.1 Multilevel paradigm

The multilevel approach was originally developed to speed-up move-based search algorithms for graph and hypergraph partitioning problems [14, 18, 19]. The expression "multilevel" comes from reducing the size of the original problem instance at different degrees (called *levels*), yielding a sequence of increasingly smaller problem instances

(multilevel).

4.1.1 Graph partitioning problem

Assume $G = (V, E)$ is a undirected graph, let $|V| = n$ and $|E| = m$ be respectively the number of vertices and edges in G. We denote edges in G either as $e \in E$ or $\{a, b\}$ where a, b is a pair of vertices in V. The k-way graph partitioning problem consists of finding a partitioning of V into k disjoint sets $\{P_1, P_2, ..., P_k\}$ that minimizes the following cost function:

$$f(x) = \sum_{i=1}^{m} w(e_i)$$

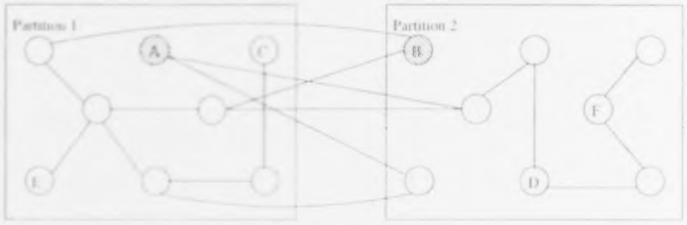
subject to:

1. $P_i \cap P_j = \emptyset$ for $i \neq j$,
2. $\frac{|V|}{ck} \leq |P_i| \leq \frac{c|V|}{k}$ for some constant $c \geq 1.0$,
3. $\cup_{i=1}^{k} P_i = V$ and
4. $$w(e_i) = \begin{cases} 1 & \text{if } e_i \text{ is a edge spanning more than one partition } P_j, \\ 0 & \text{otherwise.} \end{cases}$$

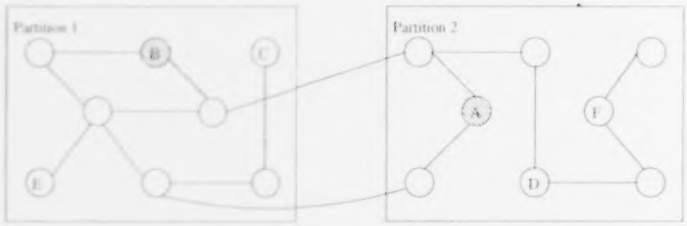
In other words, the objective of the graph partitioning problem is to find a partition of V which minimizes the number of edges spanning more than one set.

The graph partitioning problem is a well known NP-hard combinatorial optimization problem [9]. Several neighborhood search heuristics have been proposed for this problem [17, 20, 28, 31]. Typically, these search heuristics are based on neighborhoods defined by swapping pairs of vertices between partition sets. Figures 4.1 (a) and 4.1

(b) illustrate a problem instance for the 2-way graph partitioning problem. In Figure 4.1 (a), 6 edges span the two partition sets, therefore the cost of this partitioning is $f = 6$. The solution strategy consists to improve the current solution by swapping a pair of vertices between the two partition sets. The neighborhood of the solution in Figure 4.1 (a) is made of solutions that can be obtained by swapping pairs of vertices like $\{A, B\}, \{A, D\}, \{A, F\}, \{C, B\}, \{C, D\}, \{C, F\}$ and so on. Most swaps change the number of edges that span both partition sets. The swap that produces the largest improvement in the cost function is $\{A, B\}$, which is illustrated in Figure 4.1 (b). This move reduces the number of edges spanning 2 partition sets to only two. Hence, it yields a solution of cost $f = 2$.



(a) Cost 6



(b) Cost 2

Figure 4.1: Cost of the 2-way partitioning after swapping vertices A and B

4.1.2 Application of the multilevel paradigm to the graph partitioning problem

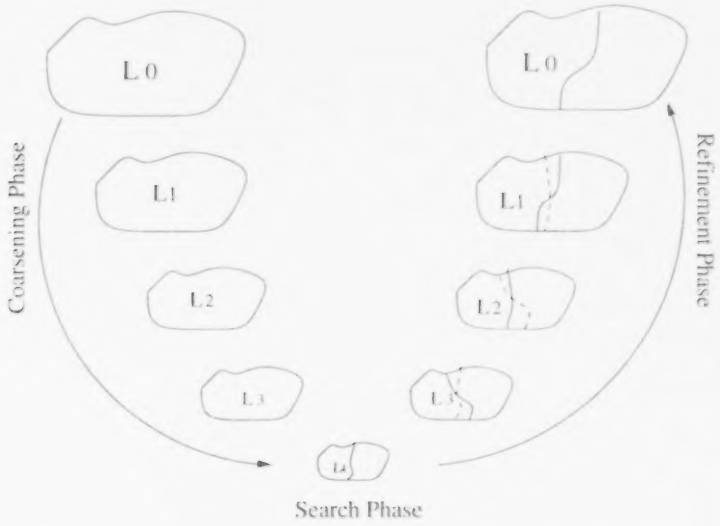


Figure 4.2: Phases of the multilevel method for the GPP

We can identify 3 important phases in multilevel algorithms, as illustrated in Figure 4.2 [19] for the graph partitioning problem:

1. *Coarsening phase:* The size of the original problem instance is recursively reduced by fixing the state of some decision variables;

2. *Search phase:* The smallest problem instance is solved using an exact or a heuristic search method;

3. *Refinement phase:* The best solution at one level is refined using a search procedure applied to the problem instance at the next less coarsened level.

We illustrate these three phases with an application of the multilevel approach to the graph partitioning problem.

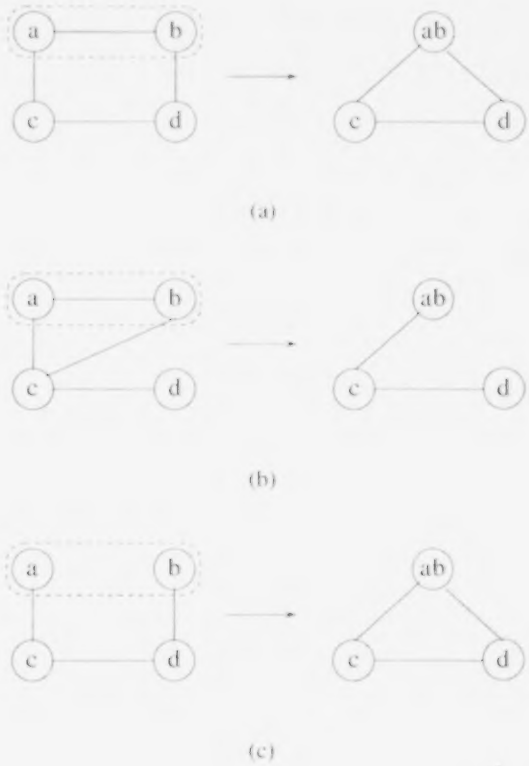


Figure 4.3: Coarsening phase

Coarsening phase: Let $G_0 = (V_0, E_0)$ be the original problem instance. The set of decision variables of G_0 consists of the vertices in V_0. The domain of each decision variable is $\{1, \ldots, k\}$, where k is the number of partition sets for the problem instance. One can reduce the number of decision variables by merging randomly selected pairs of vertices $a, b \in V_0$ into a single vertex ab. Merged vertices yield a new graph $G_1 = (V_1, E_1)$ in the following manner: if there exist an edge $\{a, b\}$ in G_0, then this edge does

not appear in G_1 (shown in Figure 4.3 (a)); if there exist edges $\{c, a\}$ and $\{c, b\}$ in G_0, they are merged into a single edge $\{c, ab\}$ in G_1 (shown in Figure 4.3 (b)); if there exist edges $\{c, a\}$ and $\{d, b\}$ in G_0, these edges become $\{c, ab\}$ and $\{d, ab\}$ in the coarsened graph G_0 (shown in Figure 4.3 (c)). As illustrated in Figure 4.2, each iteration of the coarsening phase yields a smaller graph (where the number of vertices is reduced by about half). Coarsening is recursively applied to the latest coarsened graph until the problem instance is small enough to be solved exactly or search efficiently by some heuristic method.

Notice that solutions of the coarsened graph G_1 are also solutions of the original problem instance G_0. Assigning a vertex $ab \in V_1$ to a partition set x is equivalent to assign vertices $a, b \in V_0$ to the partition set x. Denote by X_0 the solution space for G_0 and X_i the solution space of the coarsened graph G_i. Then, given the coarsening strategy as described above, the following relation holds among solution spaces of each level: $X_l \subset X_{l-1} \subset ... \subset X_1 \subset X_0$, where l is the last level.

Search phase: Performed on the very small problem instance G_l using an exact search algorithm or a neighborhood search heuristic, it speed-ups considerably the search. However, search on the restricted solution space X_l is unlikely to produce good solutions. Such good solutions may simply not exist in the solution space X_l.

Refinement phase: It is a search performed on the problem instance at levels 0 to $l-1$ using a neighborhood search heuristic. The main purpose of the search heuristic at level i is to "refine" the best solution identified by the search heuristic at level $i+1$. Let $G_i = (V_i, E_i)$ be the coarsened graph at level i. Assume the best solution at level $i+1$ is stored in an array of integers S_{i+1} of size $|V_{i+1}|$ such that entry $S_{i+1}[j]$ indicates in which partition the vertex j is assigned in the corresponding solution S_{i+1}. The first step of the refinement phase for level i consists to use the best solution of level $i+1$ as a seed to interpolate an initial solution for the search heuristic at level i. This interpolation

operation consists to split each vertex $ab_{i+1} \in V_{i+1}$ into vertices $a_i, b_i \in V_i$. Since vertices a_i and b_i in G_i have been merged during the coarsening phase to obtain vertex ab_{i+1} in G_{i+1}, we can obtain a feasible initial solution for level i by assigning vertices a_i and b_i in the solution S_i to the same partition as $ab_{i+1} \in V_{i+1}$ in the solution S_{i+1}. The next step of the refinement phase for level i consists to execute a search of the solution space X_i using the interpolated solution as starting point for this search. Search at level i refines the interpolated solution because vertices a_i and b_i are no longer always assigned to the same partition, which usually yields solutions with reduced cost in comparison with the solution of level $i + 1$ that has been used by the interpolation process. A relatively brief search is performed to refine the interpolated solution. Once the search is completed at a level i, the best solution is sent to level $i - 1$. The refinement phase is completed once the search terminates at level 0 (the level of the original problem instance).

While multilevel algorithms are fast, the quality of the approximations provided by these algorithms depends crucially on how the coarsening phase reduces the solution space [38]. This aspect of the search behavior of multilevel algorithms is illustrated in Figure 4.4. This Figure describes a 5 levels multilevel algorithm for the graph partitioning problem. In each subfigure, the x-axis represents solutions while the y-axis represents the cost of each solution. The curve symbolizes the cost function. On the left hand side of Figure 4.4, from top to bottom, the original problem (level 0) is coarsened increasingly until there is only one solution in the solution space (level 5). Suppose two vertices v_1 and $v_2 \in G_{i-1}$ at level $i - 1$ are merged to form a single vertex $v \in G_i$ at level i. Any partitioning of coarsened graph G_i is then restricted to always place both vertices v_1 and v_2 together in the same partition set (since they have been merged into a single vertex at level i). Therefore, this coarsening reduces the solution space of G_i given that solutions with v_1 and v_2 in different partitions do not exist in G_i. This exclu-

sion of some solutions with the increase level of coarsening is called the *sampling effect* of coarsening.

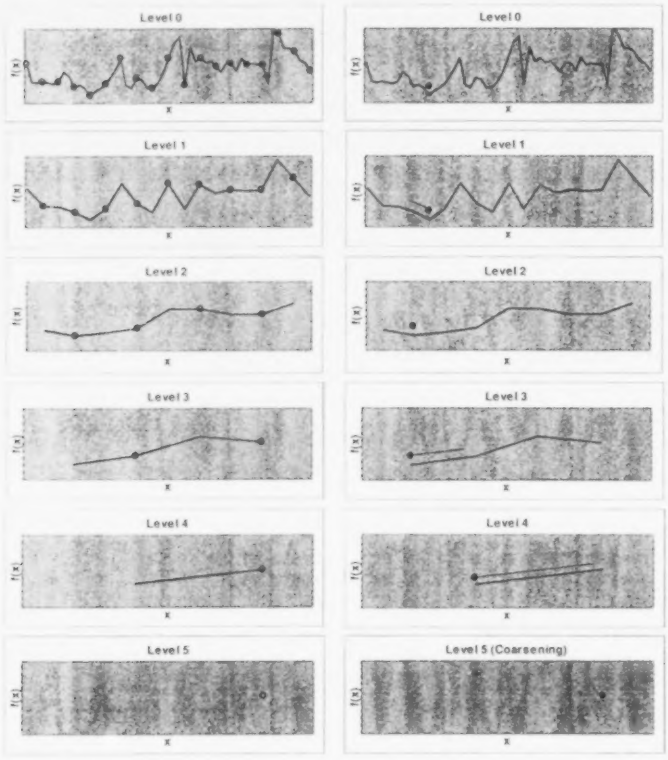


Figure 4.4: Behavior of multilevel paradigm for the graph partitioning problem (the global optimal solution is found)

Coarsening also smooth the cost function. This smoothing of the cost function removes barriers between regions of the solution space and it makes easier for search procedures to go directly toward "good" regions of solution space. In Figure 4.4, the

sampled points are circled and all intermediate values are removed to give to the next coarser version. The only solution in the coarsest problem becomes the initial solution (shown as a black dot in the bottom right hand side of the figure) although it is not an optimal solution to the overall problem. This initial solution becomes a starting solution for the less coarsened level. On the right hand side of Figure 4.5, from bottom to top, this best solution found is then sent to the next lower level recursively up to level 0.

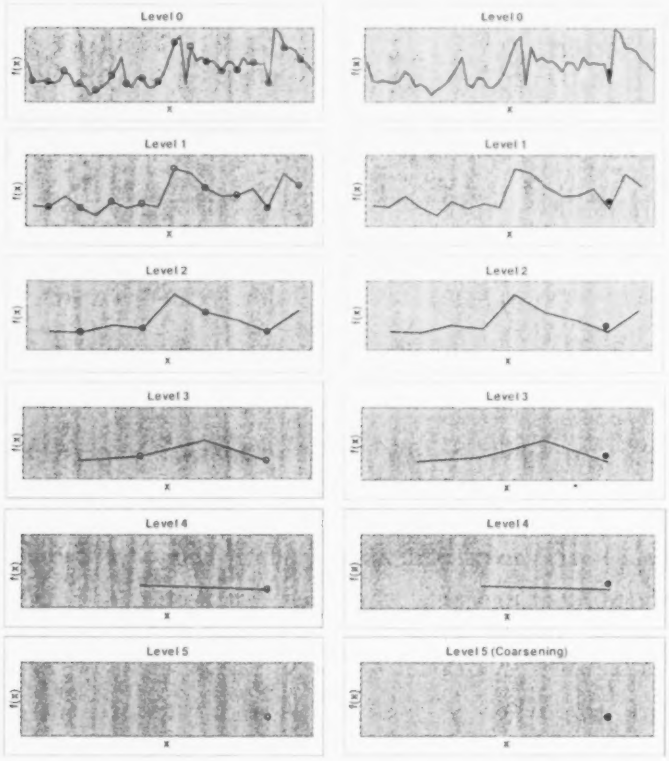


Figure 4.5: Behavior of multilevel paradigm for the graph partitioning problem (the global optimal solution is not found)

Multilevel algorithms are sensitive to the way the coarsening phase samples the solution space [37]. In Figure 4.4, the global optimal solution is found. It is possible the refinement policy will fail to find the global minimum if a different set of sampling points is picked, i.e., if another coarsening strategy is chosen. For example, Figure 4.5 gives the search behavior of multilevel algorithms for the same problem instance as the one in Figure 4.4. Since a different coarsening strategy is chosen, the only solution in the coarsest problem is not improved during the refinement phase (from the bottom to the top on the right side of Figure 4.5). So, the global optimal solution is not found in this example.

4.2 Cooperative search

The exploration of the solution space by search heuristics can be characterized as a path in the solution space (shown in Figure 4.6). Each point in this search path is a solution visited by the search heuristic. Points which are not in the search path are neighboring solutions and arcs from a visited solution to the next solution identify a move in the neighborhood of a visited solution. It comes as a natural observation that the exploration of the solution space can be decomposed along several search paths, where each search path is generated by a different search procedure. This observation has been key to the development of a parallelization strategy for neighborhood search heuristics named *parallel independent search*. In the parallel independent search approach, the solution space is decomposed implicitly, by having the search procedures start from different initial solutions or use different search parameters, or by using totally different search heuristics. It has been shown that parallel independent search not only speed up the exploration of the solution space (more solutions are visited for a same search time), but it also helps to find better solutions compared to the sequential search that visits the

same number of solutions in the solution space [16, 30].

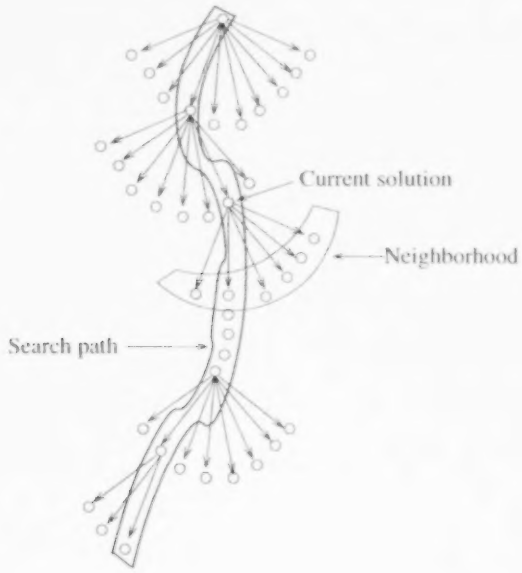


Figure 4.6: Exploration of the solution space

Cooperative search is a extension of the parallel independent search strategy. In a cooperative search algorithm, the different search procedures exchange information regarding the progress they make in the exploration of the solution space [32, 35]. This exchange of information imposes a global search strategy on top of the parallel independent search strategies. The issues regarding the global strategy in cooperative search are generally quite complex and will not be addressed in this thesis. Interested readers should consult the following papers [32, 35]. However, experimental results show that cooperation among independent search procedures helps find better solutions to combinatorial optimization problems [22, 23, 29, 33] as well as to constraint satisfaction problems [4, 15].

4.3 Multilevel cooperative search

Multilevel cooperative search is a parallel algorithm design strategy combining cooperative search and the multilevel approach. As for cooperative search, the parallelism of multilevel cooperative algorithms is based on a set of independent search procedures concurrently exploring the solution space of the same problem instance. However, unlike cooperative search, the search path performed by each independent procedure is determined by the search heuristic and a coarsening strategy defining search spaces.

A multilevel cooperative search procedure starts by coarsening the original problem instance. Each coarsened problem instance generated by the coarsening procedure is used as input for the search heuristics associated to each level. The solution space of the original problem instance as well as the solution space of each coarsened problem instance is then explored concurrently by a set of independent search procedures. Since we have $X_i \subset X_0$, exploring the solution space of the problem instance at level $i > 0$ is equivalent to searching regions of the solution space of the original problem instance defined by X_i.

4.3.1 Interaction operators for the graph partitioning problem

As mentioned before, the sampling effect of coarsening may exclude "good" solutions in "good" regions of X_0. Multilevel cooperative search uses "interaction operators" to refine the search at each level. These operators work by: 1- changing the coarsening of the problem instance at level i using solutions from level $i-1$; 2- reinitializing the search heuristic at level i using an imported solution from level $i+1$. The adaptation of the multilevel cooperative search strategy to the graph partitioning problem uses three interaction operators. Two operators, *local partitioning* and *local clustering* refine the

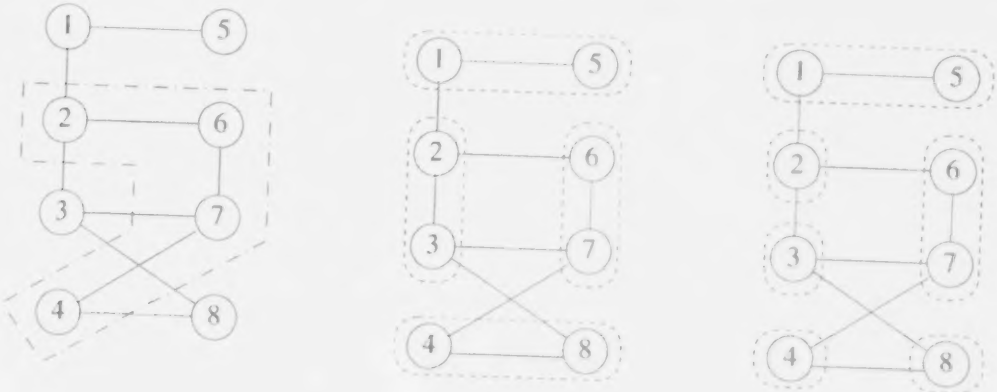
coarsening of the problem instance at each level with the help from the lower level. The *interpolation operator* has the same purpose as the interpolation operator for standard multilevel algorithms. It reinitializes the heuristic at level i based on solutions obtained from level $i + 1$.

All three operators exchange elite solutions (i.e., the current best solutions) which are discovered in the solution spaces of neighbor levels. Information exchange in multilevel cooperative search takes place only between problem instances corresponding to adjacent levels in the multilevel structure defined by coarsening. This refinement phase of multilevel cooperative search is illustrated in more details in the next three subsections through the implementation of the three operators for the graph partitioning problem.

4.3.1.1 Local partitioning

This interaction operator changes the coarsening of the problem instance at level i by splitting some of the vertices in the coarsened graph G_i, which increases the number of clustered vertices in G_i. The procedure first identifies which vertices can be splitted by comparing all the vertices of level i with an elite solution from level $i - 1$. Recall that a vertex in a coarsened graph at level i is a cluster of vertices from the original problem instance at level 0. The comparison between a vertex $v \in G_i$ and an elite solution x from level $i - 1$ aims to determine if the cluster of vertices from G_0 forming vertex v belongs to a single partition in the elite solution x. The vertices in G_i which will be split are those which do not belong to a single partition in x. The reasoning behind the partitioning operator is the following: a clustered vertex at level i that overlaps partitions in an elite solution from level $i - 1$ will prevent the search heuristic at level i from reaching the good region of the solution space identified by the elite solution from level $i - 1$. In order to allow the search heuristic to explore that region, the clustered

vertex is splitted into new vertices each belonging strictly to a single partition from the elite solution x. The search at level i can then move independently the components of the previously clustered vertex and place them into a partitioning configuration similar to the elite solution from level $i-1$. Local partitioning not only increases the number of the vertices but it also increases the size of the solution space X_i to include the "good" regions of the solution space X_{i-1}.



(a) Level 0: one elite solution

(b) Level 1: original coarsening

(c) Level 1: modified coarsening

Figure 4.7: Local partitioning operator

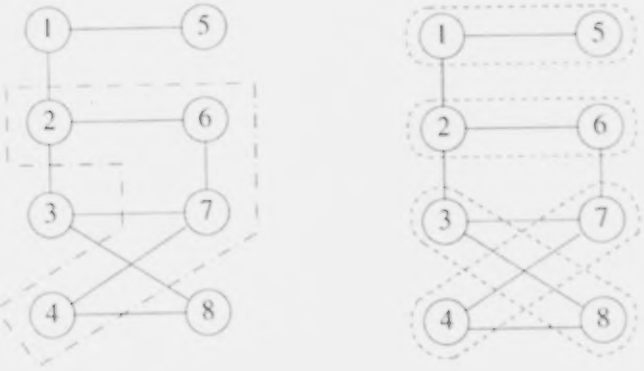
In Figure 4.7, we use a 2-way partitioning problem instance to illustrate the local partitioning operator. Vertices of G_0 are not coarsened, but each vertex of G_1 is constituted by a pair of vertices from G_0 by the coarsening procedure. Figure 4.7 (b) contains 4 clustered vertices a, b, c and $d \in V_1$ where $a = \{1, 5\}$, $b = \{2, 3\}$, $c = \{4, 8\}$ and $d = \{6, 7\}$. Let the elite solution S'_0 of level 0 be the two partitions $P_{01} = \{1, 3, 5, 8\}$ and $P_{02} = \{2, 4, 6, 7\}$ shown in Figure 4.7 (a). Local partitioning operator imports this elite solution from level 0 and check each vertex of level 1. Once the vertex of level 1 spans

more than one partition, this vertex is splitted to two vertices. As shown in Figure 4.7 (c), vertices b and c are splitted into two vertices respectively: $b_1 = \{2\}$, $b_2 = \{3\}$ and $c_1 = \{4\}$, $c_2 = \{8\}$.

4.3.1.2 Local clustering

Local clustering is a second operator which redefines the coarsening of coarsened graphs G_i. Similar to the local partitioning, the local clustering imports an elite solution from level $i-1$. Local clustering consists scanning the partition sets of an elite solution x from level $i-1$ and merging a pair of vertices $u, v \in G_{i-1}$ if they belong to the same partition in x and they are adjacent vertices in G_{i-1}. Vertices defining G_i before the local clustering are removed and replaced by those created by the local clustering operation. Local clustering aggregates into a single vertex of G_i those vertices of G_0 in the same partition of an elite solution of level $i-1$. The rational is that vertices belonging to the same partition in an elite solution can be considered as settled, we make them a single vertex in G_i where they can be moved together into the partition sets of G_i. Usually, this operator reduces the number of vertices of the coarsened graph G_i , thereby reducing the size of the solution space. Local clustering makes the search retreat from uninteresting regions of the solution space.

We give an example that explains how the local clustering operator works. Figure 4.8 represents the same 2-way partitioning problem instance used to describe local partitioning operator. In Figure 4.8 (a), the elite solution S'_0 at level 0 includes two partition sets: $P_{01} = \{1, 3, 5, 8\}$ and $P_{02} = \{2, 4, 6, 7\}$. As shown in Figure 4.8 (b), since vertices $\{1\}$ and $\{5\}$ of level 0 are in the same partition, and also are connected by the same edge, they therefore become a new vertex $\{1, 5\}$ of level 1. Similar to vertices $\{3\}$ and $\{8\}$, $\{2\}$ and $\{6\}$, $\{4\}$ and $\{7\}$.



(a) Level 0: one elite solution

(b) Level 1: re-coarsening

Figure 4.8: Local clustering operator

4.3.1.3 Interpolation

The interpolation operator reinitializes the search of level i using an elite solution from level $i+1$ as initial solution for the search. One of the motivation for multilevel search is to use the small solution spaces to perform an extensive exploration of the solution space of the original problem instance. This extensive search identifies regions of the solution space that might be worth a more intensive search. The interpolation operator implements this transition from extensive to more intensive searches. It uses a solution from a coarsened problem instance to initialize a search in the solution space of a less coarsened problem instance. This search has more flexibility to move vertices between partition sets, it can then refine the interpolated elite solution.

4.4 Summary

In summary, multilevel cooperative search algorithms use coarsening to identify "good" regions of the solution space that can be explored in parallel by a set of independent search heuristics. Because the initial coarsening samples the solution space randomly, this may result in coarsened problem instances that have a poor representativity of "good" regions of the solution space. Operators like local partitioning and local clustering use the knowledge accumulated by exploring the different search spaces to refine the original coarsening. Local partitioning operator undoes the coarsening of decision variables if they appear to have different values in the elite solutions found by the search heuristics. On the other hand, local clustering operator merges decision variables that appear to often have the same values in good solutions identified by search heuristics. Improving the coarsening at level i helps search heuristics to identify better solutions, which in turn can be used to improve the coarsening of the problem instances in adjacent levels to level i. In addition, multilevel cooperative search algorithms use interpolation operator to identify "good" regions of the solution space by reinitializing independent search heuristics.

Chapter 5

Multilevel Cooperative Tabu Search (MCTS) Algorithm for the CMND Problem

This chapter describes our multilevel cooperative tabu search (MCTS) algorithm for the capacitated multicommodity network design (CMND) problem. This algorithm uses the cycle-based tabu search algorithm of Chapter 3 to search the coarsened problem instances corresponding to each level of the multilevel structure. The main contributions in this algorithm are the design of a coarsening strategy for the CMND problem, the definition of interaction operators to restart search and redefine new coarsened problem instances, and modifications to the cycle-based tabu search algorithm in order to adapt this search procedure to the constraints imposed by the multilevel algorithm.

In this chapter, we first describe our coarsening procedure for the CMND problem. Next, our three interaction operators are described. We then identify the modifications we have made to the the cycle-based tabu search procedure. Finally, we show how all

the pieces of the puzzle combine to form our multilevel cooperative search algorithm.

5.1 Coarsening phase

We have considered two ways to coarsen the graph representing a network design problem. First, we tried to coarsen network design graphs by merging pairs of vertices from the network. This approach presented some serious difficulties in the context of graphs representing network design problems. Unlike graphs for the GPP, network design graphs are weighted directed graphs. Each time an arc y_{ij} is removed from the graph, it destroys paths in the network as well as the information associated with the corresponding arc such as its fixed cost, routing cost per commodity, flow capacity and the number of units of flow in the current solution. Therefore, changes in network design graphs resulting from merging pairs of vertices presented serious challenges in terms of conserving the information associated to deleted arcs into the new coarsened network design graphs. We couldn't come with an easy approach to handle this problem, therefore, we have abandoned this coarsening strategy in the context of the present research.

We have chosen another coarsening strategy which consists to fix the status of some arcs as either *fixed-open* or *fixed-closed*. Decision variables (arcs) which are fixed-open or fixed-closed become constants, i.e., their state cannot be modified by a search heuristic exploring the solution space. For example, flow cannot be deviated on an arc that is fixed-closed, therefore no cycle can include such an arc. Similarly, a fixed-open arc cannot be closed. Fixing arcs has the effect of coarsening the original problem instance by reducing the number of decision variables, i.e., arcs that can swap in and out the design. However, since arcs still exist in the network, the coarsened graph keeps all the information on each arc needed to compute the cost function and determine feasibility of each solution.

5.1.1 Selecting arcs with fixed status

In order to coarsen the original network design problem instance into smaller problem instances, we need to determine the number of levels required and the number of arcs with a fixed status in each coarsened problem instance. We also need to decide which arcs are given a fixed status. The number of levels as well as the number of arcs with fixed status are dependent on the size of the original network design problem instance. Consequently, these values are determined during the calibration phase. We will describe how these parameters are determined in Chapter 6.

In the present section, we describe how we select arcs that are given a fixed status. We have investigated two approaches. Both aimed to determine the fixed status of arcs based on their likelihood to be opened or closed in good solutions. The two approaches investigated are: the Fixed Cost Method (FCM) and the Frequency Method (FM).

5.1.1.1 Fixed Cost Method

In the fixed cost method, arcs are selected to be fixed-open or fixed-closed based on their $\frac{\text{fixed cost}}{\text{capacity}}$ ratio (FC ratio). An arc with a small FC ratio means the arc has a small fixed cost compared to its capacity. On the other hand, an arc with large FC ratio means the arc has a big fixed cost comparing to its capacity. Introducing an arc with a low fixed cost and high capacity does not increase substantially the overall cost of the design while adding capacity to carry a large number of units of flow. On the other hand, adding arcs with a high fixed cost and low capacity in a design increases substantially the overall cost of the design while allowing to carry only a few units of flow. Hence, we think arcs with a small FC ratio are more likely to be opened in good solutions and arcs with a big FC ratio are more likely to be closed in good solutions.

Algorithm 2 shows our fixed cost method as a coarsening strategy. In this method,

arcs are chosen to be fixed-open or fixed-closed by a random function which is biased by the arcs FC ratio. The smaller the FC ratio of an arc, the more likely this arc will be selected to be fixed-open. On the other hand, the bigger the FC ratio of an arc, the more likely the status of the arc will be fixed-closed. To implement this biased random selection, random numbers are selected uniformly in the interval $[0, 1]$. We partition the interval $[0, 1]$ into n sub-intervals, where n is the number of arcs in the network. Each sub-interval j corresponds to an arc j in the network. To bias the arc selection based on the FC ratio, the length of sub-interval j is made proportional to the FC ratio of arc j. For fixed-open arcs, the smaller the ratio of an arc, the longer is the corresponding sub-interval for this arc. However, for fixed-closed arcs, the bigger the ratio of an arc, the longer the sub-interval corresponding to this arc. An arc j is selected to be fixed-open or fixed-closed when the number return by $uniform(0..1)$ falls in the sub-interval j. The longer the sub-interval corresponding to an arc is, the more likely the random number returned by the function $uniform(0..1)$ falls in this sub-interval.

5.1.1.2 Frequency Method

The frequency method is a second approach to set the status of arcs as fixed-open or fixed-closed. As for the fixed cost method, the frequency method seeks to determine the status of arcs based on the likelihood these arcs will have the same status in good solutions. Assuming fixed-open arcs will appear in "good" solutions and fixed-closed arcs will not, we fix the status of arcs during coarsening according to how often arcs appear as opened or closed in "good" solutions. If an arc is frequently opened in "good" solutions, coarsening try to set it to be fixed-open; if an arc is frequently closed in "good" solutions, coarsening try to set it to be fixed-closed.

The procedure to determine the fix status of arcs using the frequency method is relatively simple. A sequential cycle-based tabu search procedure is run for a pre-

determined number of iterations. This cycle-based tabu search procedure builds a set of *elite solutions* ES. The solutions in the elite set are the best solutions ever found by the search procedure. The cardinality of the elite set is a pre-defined parameter. The first $|ES|$ iterations of the sequential search procedure fills the elite set. After the first $|ES|$ iterations, if the current solution is better than the worst solution in the elite set, then this worst solution is replaced by the current solution.

During the execution of the cycle-based tabu search, a current solution is considered a *promising solution* if its cost is better than the worst solution in the elite set or close to it by a pre-defined percentage. The promising solutions set is representative of good solutions for a problem instance. Each time a promising solution is identified during the cycle-based tabu search, a vector called *frequency memory* is updated. This vector records how often arcs are opened or closed in promising solutions. Each entry in the frequency memory corresponds to an arc in the network. If an arc is opened in a promising solution, the corresponding entry in the frequency memory is increased by 1. If an arc is closed in the promising solution, the corresponding entry in the frequency memory is decreased by 1.

In Algorithm 3, once the cycle-based tabu search is completed, arcs in the frequency vector are sorted in decreasing order of their frequency. The top $NbFixingOpen$ arcs are set to be fixed-open; the last $NbFixingClose$ arcs are set to be fixed-closed. The specific number of iterations of the cycle-based tabu search, the cardinality of the elite set and a percentage to determine the promising solution are values obtained during the parameter calibration of the algorithm.

Algorithm 2 Coarsening Algorithm of FC Method

$CoarsenLevel(G, NbFixedOpen, NbFixedClose, i)$

Input: G is the network

NbFixedOpen is the number of arcs that are Fixed-Open

NbFixedClose is the number of arcs that are Fixed-Close

i is the index of the level

Output: Y_i, an array of size n such that $Y_i[j]$ = FixedOpen, FixedClose or Undetermined.

int k=1;

each element of Y_i is initialized as Undetermined;

while($k \leq NbFixedOpen$)

u = uniform(0..1);

j = position_FixedOpen(u);

if($Y_i[j]$ == Undetermined) **then**

$Y_i[j]$ = FixedOpen;

k++;

k=1;

while($k \leq NbFixedClose$)

u = uniform(0..1);

j = position_FixedClose(u);

if($Y_i[j]$ == Undetermined) **then**

$Y_i[j]$ = FixedClose;

k++;

5.1.2 Coarsening procedure

The coarsening strategy for the CMND problem is the following. Assume the original CMND problem instance is represented by a graph $G = (N, A)$ where $|A| = n$. The coarsening phase for the CMND problem generates l coarsened network design problems by fixing the status of $NbFixedOpen_i + NbFixedClose_i$ arcs of G for each level i. Let $n_i = n - (NbFixedOpen_i + NbFixedClose_i)$ be the number of arcs that can move in and out the design at level i. The coarsening procedure is such that $n_i < n_{i-1}$. Algorithm 2 summarizes a coarsening procedure based on selecting fixed arcs using the fixed cost method.

In Algorithm 2, the variables $NbFixedOpen$ and $NbFixedClose$ denote the predefined number of arcs that are fixed-open or fixed-closed for level i. The variable Y denotes an array of n integers. This array is used to store the fix status of arcs, $Y[j]$ states the fixing status of arc j. Initially, the status of each arc is set as "Undetermined", which means the status of each arc can be changed by the cycle-based tabu search heuristic. The first **while** loop select arcs that will have a fixed-open status. The function $uniform(0..1)$ returns a position u on the interval $[0, 1]$. The procedure $position_FixedOpen(u)$ computes the corresponding sub-interval for position u. The procedure $position_FixedOpen(u)$ takes as input the $\frac{\text{fixed cost}}{\text{capacity}}$ ratio r_j of each arc j. We choose R to be the maximum among the ratios r_j. For fixed-open arc, we want to give large sub-intervals to arcs with small ratios $\frac{\text{fixed cost}}{\text{capacity}}$. Let $s_j = R - r_j$, we then have s_j larger when the ratio r_j of arc j is smaller. The normalization factor s'_j determines the percentage of the interval $[0, 1]$ occupied by arc j. Sub-intervals are distributed on the interval $[0, 1]$ in increasing order of the integer label of the arcs in the network. The arc corresponding to the sub-interval holding the position u is returned by the procedure $position_FixedOpen(u)$. The second **while** loop in Algorithm 2 selects

Algorithm 3 Coarsening Algorithm of Frequency Method

$CoarsenLevel(n, V_i, NbFixedOpen, NbFixedClose)$

Input: n is the number of arcs in network

V_i is the frequency memory in level i

NbFixedOpen is the number of fixed-open arcs

NbFixedClose is the number of fixed-closed arcs

Output: Y_i is the fixing status of arcs.

int $B[0..n-1]$; // keep arc's original position

// Initialization

for $(k = 0; k < n; k++)$ $B[k] = k$;

// 1. Sort the frequency memory in decreasing order

for $(k = 0; k < n; k++)$

  for $(j = k+1; j < n; j++)$

    if $(V_i[j] > V_i[k])$ swap($V_i[j]$, $V_i[k]$); swap($B[j]$, $B[k]$);

// 2. Reset the fixed status of arcs in Y_i

for $(k = 0; k < n; k++)$ // Fixed-open arcs

  if $(k < NbFixedOpen)$ $Y_i[B[k]]$ = FixedOpen;

  else $Y_i[B[k]]$ = Undetermined;

for $(k = n-1; k \geq 0; k--)$ // Fixed-closed arcs

  if $(k \geq n - NbFixedClose)$ $Y_i[B[k]]$ = FixedClose;

// 3. Clear the frequency memory V_i

for $(k = 0; k < n; k++)$ $V_i[k] = 0$;

the arcs that receive a fixed-closed status. The procedure $position_FixedClose(u)$ obtains the arc corresponding to position u on the interval $[0, 1]$. While the probability of selecting arc j to be fixed-open is inversely proportional to its $\frac{\text{fixed cost}}{\text{capacity}}$ ratio, the procedure $position_FixedClose(u)$ selects arcs to be fixed-closed proportionally to their ratio $\frac{\text{fixed cost}}{\text{capacity}}$. Procedures $position_FixedOpen(u)$ and $position_FixedClose(u)$ are described respectively in Figures 5.1 and 5.2.

$position_FixedOpen(rn)$

Input: rn is a random number obtained from a uniform function

Output: the arc that occupies the sub-interval where rn falls.

Let r_j be the ratio $\frac{\text{fixed cost}}{\text{capacity}}$ of arc j and $R = \max\{r_j\}$;

Let $s_j = R - r_j$, $T = \sum_{j=1}^{n} s_j$;

Normalize the values of s_j: $s'_j = \frac{s_j}{T}$. Therefore $\sum_{j=1}^{n} s'_j = 1$;

Each arc j occupies $s'_j \times 100\%$ of the interval [0, 1];

If rn falls in the sub-interval of s'_j, the arc j is chosen to be fixed-open. That is, arc j is returned.

Figure 5.1: Procedure $position_FixedOpen(rn)$

$position_FixedClose(rn)$

Input: rn is a random number created by uniform function

Output: the arc that occupies the sub-interval where rn falls.

Let r_j be the ratio $\frac{\text{fixed cost}}{\text{capacity}}$ of arc j and Sum = $\sum_{j=1}^{n} r_j$;

Let $s'_j = \frac{r_j}{Sum}$. Therefore $\sum_{j=1}^{n} s'_j = 1$;

Each arc j occupies $s'_j \times 100\%$ of the interval [0, 1];

If rn falls in the sub-interval of s'_j, the arc j is chosen to be fixed-closed. That is, arc j is returned.

Figure 5.2: Procedure $position_FixedClose(rn)$

The solution space X_i associated to the network design problem at level i is obviously a subset of X_0, the solution space of the original network design problem. However, though $n_i < n_{i-1}$, the set of fixed-open arcs at level i is not a subset of the fixed-open arcs at level $i-1$. Similarly, the set of fixed-closed arcs at level i is not a subset of the fixed-closed arcs at level $i-1$. Consequently, the relation $X_l \subset X_{l-1} \subset ... \subset X_1 \subset X_0$ is not directly satisfied using the above coarsening strategy. Interaction operators are designed to address this problem.

5.2 Interaction operators

We are using three interation operators in our multilevel cooperative search algorithm: re-coarsening, interpolation and reverse interpolation.

5.2.1 Re-coarsening operator

The purpose of the re-coarsening operation is to redefine the set of arcs with a fixed status at a given level i. These changes are based on a set of promising solutions at level $i-1$ identified by the exploration of the solution space at that level. The re-coarsening operator then uses the content of frequency memory V_{i-1} (defined in Section 5.1.1.2) to decide which design arc at level i will have a fixed status.

The re-coarsening operator sorts the frequency memory V_{i-1} in decreasing order. The fixed status of arcs at level i is changed based on the arcs with the highest frequency or the lowest frequency in V_{i-1}. The re-coarsening operator uses the numbers $NbFixedOpen$ and $NbFixedClose$ associated to level i to set the number of arcs that should have a status fixed-closed or fixed-open. Entries of $B[k]$ indicate the arc $B[k]$ has the k-th highest frequency. At the end of the re-coarsening procedure, after resetting the fixed status of arcs at level i using the frequency memory V_{i-1}, this frequency memory V_{i-1} is cleared and will be updated during the following cycle-based tabu search at level $i-1$.

In the re-coarsening operator, the way to select arcs to be fixed is similar to the frequency method of the coarsening phase (Algorithm 3). The only difference is that the frequency method obtained the fixed status of arcs of level i using the frequency memory V_i generated at the current level i, while the re-coarsening operator uses the frequency memory V_{i-1} at lower level $i-1$.

5.2.2 Interpolation operator

The interpolation operation is initiated at level i by requesting an elite solution $EliteSolutionG_{i+1}$ from level $i+1$. This elite solution come from an elite set generated by the cycle-based tabu search at level $i+1$. $EliteSolutionG_{i+1}$ should be the

best elite solution which had never been sent **down** from level $i+1$ to level i yet. If all the solutions in the current elite set at level $i+1$ have already been sent to level i, then the earliest one ever sent by level $i+1$ is resent to level i.

The status of arcs in solutions obtained from level $i+1$ might conflict with the fixed status of some arcs at level i. There are two cases to consider. One is when an elite solution at level $i+1$ with arc j opened is not a feasible solution in the restricted solution space of level i where arc j is set as fixed-closed. In this case, we resolve the conflict by removing the fixed status of the conflicting arc at level i, i.e., we modify the fixed status of arc j from fixed-closed to undermined. In the second case, arc j in $EliteSolutionG_{i+1}$ is closed and arc j is fixed-open at level i. This case is not a problem because a fixed-open arc j may be closed during the trim phase which closes opened arcs with no flow after solving the CMCF problem using CPLEX. After modifying the fixed status of arcs at level i according to the elite solution from level $i+1$, we have a new network design problem at level i. The solution space of this new problem now includes $EliteSolutionG_{i+1}$ as a feasible solution. This elite solution becomes the initial solution of the cycle-based tabu search of level i.

Figure 5.3 describes the interpolation operator. The fixed status $Y_i[j]$ of a fixed-closed arc j at level i is changed to be undetermined ($Y_i[j] = Undetermined$) if arc j is open in $EliteSolutionG_{i+1}$. This is to make sure that the search in level i can proceed freely in the region identified by $EliteSolutionG_{i+1}$. Then, the elite solution $EliteSolutionG_{i+1}$ is used as the initial solution of the cycle-based tabu search at level i. Removing the fixed status of arcs at a given level increases the size of the solution space at level i. This also increases the number of available moves (cycles) in the neighborhood of solutions at level i.

$Interpolation(Y_{i+1}, EliteSetG_{i+1})$

Input: Y_i is the fixing status of arcs in level i

$EliteSetG_{i+1}$ is the elite set of level $i+1$

Output: Y_i

Get an elite solution $EliteSolutiouG_{i+1}$ in the elite set $EliteSetG_{i+1}$ from level $i+1$;

for ($j = 1; j \leq n; j++$)

if ($Y_i[j]$ == FixedClose AND $EliteSolutionG_{i+1}[j]$ == open)

$Y_i[j]$:= Undetermined;

Set $EliteSolutionG_{i+1}$ to be the initial solution of tabu search process in level i;

Figure 5.3: Interpolation operator

5.2.3 Reverse interpolation operator

The reverse interpolation operator is similar to the interpolation operator except that elite solutions are propagated **up** from level $i-1$ to level i. As for the interpolation operator, the reverse interpolation operation is initiated at level i by requesting an elite solution from level $i-1$. This elite solution should be the best elite solution never yet sent **up** from level $i-1$ to level i. If all the solutions in the current elite set at level $i-1$ have already all been sent to level i, then the earliest one ever sent by level $i-1$ is resent to level i. Conflicting status between arcs from elite solutions and fixed status at level i are handled in the similar manner as for the interpolation operator. Figure 5.4 describes the reverse interpolation operator.

$ReverseInterpolation(Y_{i-1}, EliteSolution_{i-1})$

Input: Y_i is arc fixed status of level i

$EliteSetG_{i-1}$ is the elite set of level $i-1$

Output: Y_i

Get an elite solution $EliteSolutionG_{i-1}$ in $EliteSetG_{i-1}$ from level $i-1$

for ($j = 1; j \leq n; j++$)

if ($Y_i[j]$ == FixedClose and $EliteSolutionG_{i-1}[j] ==$ open)

$Y_i[j]$ = Undetermined;

Set $EliteSolutionG_{i-1}$ to be the initial solution of tabu search process in level i;

Figure 5.4: Reverse interpolation operator

5.3 Modified cycle-based tabu search

The fixed status of decision variables impacts the neighborhood of the cycle-based tabu search method, particularly those arcs with a fixed-closed status. Remember that the neighborhood of the current solution in the cycle-based tabu search method is composed of cycles on which flow can be deviated from one path to another one. If any of the two paths in a cycle contains an arc with a fixed-closed status, then flow cannot be deviated in this cycle, the corresponding cycle cannot be used to generate a neighbor to the current solution. Due to the fixed status of some decision variables, we have adapted the cycle-based tabu search procedure such that these cycles are not computed.

Algorithm 4 ModifiedCycleBasedTabu ($CurrentSolutionG_i$, $EliteSetG_i$)

Input: $CurrentSolutionG_i$

$EliteSetG_i$

i is the index of levels

Output: $BestSolutionCost_i$

Get $CurrentSolutionCost_i$ and $BestSolutionCost_i$;

while (a stopping criterion is not satisfied)

1. **for each arc** j

   if $Y_i[j] = FixedClose$

   Delete arc j **from the network;**

2. Determine $\Gamma(\overline{y})$ and C for the current solution $\overline{y}$;

3. **for** each $\gamma \in \Gamma(\overline{y})$

   Compute the γ**_residual network;**

   Determine $C(\gamma)$;

   for each arc $j \in C(\gamma)$

   Find the associated "low cost" cycle by using the network optimization procedure;

   Find the lowest cost cycle over all $j \in C(\gamma), \gamma \in \Gamma(\overline{y})$;

4. Move to the new solution by opening or closing corresponding arcs;

5. Evaluate the solution value of the new solution by solving the associated CMCF problem using CPLEX;

6. Trim, i.e., close opened arcs with no flow;

7. Set the tabu status to each complemented arc;

8. **if** the solution is not feasible

   Execute a *restoration* phase;

9. **if** the current level is level 0 or level 1

   if $(\frac{CurrentSolutionCost_i - BestSolutionCost_i}{BestSolutionCost_i} \leq IntensGap)$

   Perform an $intensification$ phase;

10. **if** $\boldsymbol{CurrentSolutionCost_i < WorstEliteSolutionCost_i}$ **in** $\boldsymbol{EliteSetG_i}$

    Replace $\boldsymbol{WorstEliteSolutionG_i}$ **in** $\boldsymbol{EliteSetG_i}$ **by** $\boldsymbol{CurrentSolutionG_i}$;

11. **if** $\frac{CurrentSolutionCost_i - WorstEliteSolutionCost_i}{WorstEliteSolutionCosti} \leq FrequencyGap$

    Update the frequency memory V_i;

12. **if** $CurrentSolutionCost_i < BestSolutionCost_i$

    $BestSolutionG_i := CurrentSolutionG_i$;

The modified cycle-based tabu search is described in Algorithm 5.3. Before initiating a search with the cycle-based tabu search, arcs with fixed-closed status are removed from the data structure $CurrentSolution G_i$ which stores the coarsened network design at level i (see line 1 in Algorithm 5.3). Fixed-closed arcs are removed by setting their capacities to 0 such that those fixed-closed arcs don't have capacity to deviate flow with a value γ. In line 3, $\mathcal{C}(\gamma)$ is the set of arcs capable to deviate flow a value γ. Cycles are computed for each arc in $\mathcal{C}(\gamma)$. Eliminating fixed-closed arcs from the network design reduces the cardinality of $\mathcal{C}(\gamma)$ and speeds up computation of the neighborhood of most solutions visited by the tabu search procedure. Notice that eliminating arcs also helps to speed-up the computation of the γ-residual network, which is why we have emphasized this step in the modified cycle-based algorithm. In addition, we know that in the γ-residual network, there are at most two arcs $(i,j)^+$ and $(j,i)^-$ which can replace each arc in the current network. Arc $(i,j)^+$ means to add γ more flows on arc (i,j) while arc $(j,i)^-$ means to remove γ flows on arc (i,j). Hence, since fixed-open arcs can not be closed in the cycle-based tabu search, in the γ-residual network, we eliminate $(j,i)^-$ on the γ-residual network if arc (i,j) is fixed-open such that arc (i,j) can not be closed.

In our multilevel cooperative search algorithm, the intensification phases of the cycle-based tabu search procedure are only performed at levels 0 and 1. The **if** statement on line 9 in Algorithm 5.3 enforces this condition. The main purpose of the intensification phase is to refine the search in promising regions of the solution space. We thought the computational cost of these intensification phases was not worthy when the exploration of the solution space mainly aims at discovering promising regions of the solution space. This is often the case when searching the solution space of coarsened problem instances. For coarsened instances, good solutions may be left out, therefore multilevel cooperative search use them to guide the search in less coarsened problem instances.

The interaction operators are based on elite solutions and frequency memories. Information stored in the frequency memories and elite sets is collected during the search performed by the cycle-based tabu search. Thus, we add steps 10 and 11 to the cycle-based tabu search procedure to collect information. In step 10, the modified cycle-based tabu search stores elite solutions by replacing the worst elite solution in the elite set $EliteSetG_i$ by the current solution if the current solution is better than this worst elite solution in the elite set. Step 11 describes the update of the frequency memory V_i. This update is performed once the current solution is better than the worst solution in the elite set $EliteSetG_i$ or close to it by a predefined percentage $FreqencyGap$.

5.4 Multilevel cooperative tabu search algorithm (MCTS)

In this section we describe how the coarsening phase, the interaction operators and modified cycle-based tabu search combine together to form the Multilevel Cooperative Tabu Search (MCTS) algorithm. MCTS is a parallel search algorithm, therefore, a MCTS program first reads the graph description of original network design problem instance and then spawns a number of processes equal to the number of levels. Each process i (level i), as shown in Algorithm 5.4, computes an initial solution D_i by solving the CMCF problem with all arcs opened. This initial solution D_i is placed into the elite set $EliteSetG_i$. Next, each process i performs its coarsening phase based on the original network design problem instance. The number of arcs that get a fixed status depends on level i. Then, the frequency memory V_i is initialized by having each entry set to 0.

The **while** loop of each process i controls the interaction operators and the modified cycle-based tabu search procedure. It first calls the cycle-based tabu search procedure (line 5(a)). This procedure is run for a pre-defined number of iterations. In the first iteration of the **while** loop, the modified cycle-based tabu search procedure generates a set

of elite solutions and a frequency memory for the process at level i. Subsequently, for the other iterations of the **while** loop, the modified cycle-based tabu search procedure updates these two data structures. Once the modified cycle-based tabu search procedure has completed a pre-defined number of iterations, it returns control to process i, where the interpolation operator is initiated if level i is not the highest (line 5(b)). The interpolation operator brings to level i an elite solution from level $i + 1$. On line 5(c), the elite solution from level $i + 1$ is used to restart the modified cycle-based tabu search in a region of the solution space specified by the elite solution. The solution from level $i + 1$ is an elite solution, the region of the solution space identified by this solution is likely to contain good solutions. This is why a new sequence of tabu search iterations is performed after the interpolation.

On line 5(d), the re-coarsening operator is applied. As the search in the solution space proceeds at each level, better solutions are found about the original problem instance. The number of times arcs are found opened or closed in those good solutions is recorded in the frequency memory of each level. Based on the frequency memory of level $i - 1$, re-coarsening operator redefines the coarsening at level i by changing the set of arcs with a fixed status at level i. Arcs that have been often opened in promising solutions at level $i - 1$ become fixed-open at level i through the re-coarsening operator. On the other hand, arcs frequently closed in promising solutions at level $i - 1$ become fixed-closed at level i. Having arcs fixed-closed at level i is equivalent to remove them from network designs while those that are fixed-open appear in all network designs of level i. Therefore, re-coarsening defines new network design problem instances. These problem instances have some decision variables already set-up with the likely values they will have in good solutions of the original problem instance. In this way, a search at level i after re-coarsening is likely to be the exploration of good regions of the solution space of the original problem. Therefore, following the re-coarsening operation,

a sequence of cycle-based tabu search iterations is performed (line 5(e)) on this new problem instance.

The last step of the **while** loop is a call to the reverse interpolation operator (line 5(f)). The **while** loop iterates until the stopping criterion is satisfied. The stopping criterion is based on the number of iterations executed by the modified cycle-based tabu search procedure. Each time the cycle-based procedure is called, we add the number of iterations executed. Once a a predefined total number of the modified cycle-based tabu iterations is reached, process i returns its best solution to the master program. Once each process has returned, the MCTS algorithm is terminated.

Note that each process actives interaction operators once it finished its own fixed number of iterations of cycle-based tabu search. That is, the process are not synchronized with each other.

Algorithm 5 $Process_i$ of MCTS (network, L)

Input: $network$ is a directed graph;

L is the number of levels;

Output: the cost of the best design.

1. Generate a initial network design D_i by solving the CMCF problem with all arcs opened;
2. $D_i \rightarrow EliteSetG_i$;
3. Coarsen the network to generate the fixing status Y_i;
4. Initialize the frequency memory V_i with each entry 0;
5. **while** (Stop criterion is not satisfied) // main loop
   (a) $ModifiedCycleBasedTabu(D_i, EliteSetG_i)$; // an modified cycle-based tabu search
   (b) **if** ($i \neq L-1$) // interpolation operator
   Get $EliteSolutionG_{i+1}$ from $EliteSetG_{i+1}$ at level $i+1$;
   Update the fixed status Y_i;
   $D_i := EliteSolutionG_{i+1}$;
   (c) $ModifiedCycleBasedTabu(D_i, EliteSetG_i)$; // an modified cycle-based tabu search
   (d) **if** ($i \neq 0$) // re-coarsening operator
   Get the frequency memory V_{i-1} from level $i-1$ and sort it in decreasing order;
   Set the first NbFixedOpen vectors to be fixed-open and the last NbFixed-Close vectors to be fixed-close;
   Clear the frequency memory V_{i-1};
   (e) $ModifiedCycleBasedTabu(D_i, EliteSetG_i)$; // an modified cycle-based tabu search
   (f) **if** ($i \neq 0$) // reverse interpolation operator
   Get $ElitSolutionG_{i-1}$ from $EliteSetG_{i-1}$ at level $i-1$;
   Update the fixed status Y_i;
   $D_i := EliteSolutionG_{i-1}$;
6. Output $BestSolutionCost_i$;

Chapter 6

Performance Analysis of MCTS

In this chapter we describe how the different parameters of our algorithm have been calibrated, our experimental setting and the tests that have been conducted to validate this algorithm. We conclude with an interpretation of the experimental results and a qualitative analysis of the search behavior of the algorithm. The latest analysis illustrates furthermore how our algorithm works by focusing on aspects of the search behavior influenced by coarsening and the interaction operators. This analysis identifies potential improvements in the design of the algorithm and in the fine tuning of the search parameters in order to improve the performance of the heuristic.

6.1 Calibration phase

Like any other search heuristic, some parameters of our multilevel algorithm had to be calibrated using tests. We have chosen eight problem instances to realize the calibration tests. The choices have been made mainly considering the size of each problem instance, i.e. the number of nodes, arcs and commodities. The size of problem in-

stances is one key factor determining the values associated to search parameters. We have chosen eight problem instances which are neither too large nor too small. They are all with 20 nodes, 300 design arcs, different number of commodities (40 and 200) and various combinations of ratios of fixed cost to routing cost and capacity to demand. The following parameters were calibrated using these problem instances:

1. Number of levels;
2. Coarsening factors, that is, the size of the fixed-open and fixed-closed arcs at each level;
3. Number of tabu iterations between activations of the interaction operators;
4. Dimension of the elite solutions set;
5. Conditions for updating of the frequency memory;
6. Selection of initial arcs to be fixed.

In order to limit the effort spent on calibration, we restrict the tested values for each parameter by a careful reasoning about which values are the most appropriated. During the calibration phase, each calibrated parameter is tested using this restricted set of values while keeping all the other calibrated parameters constant. The calibration phase ended by selecting a unique combination of values for all parameters which provides the best solutions on the largest number of problem instances. This single set of values is used during the experimentation phase.

6.1.1 Number of levels and coarsening factors

The coarsening factors set the number of fixed arcs at each level. The coarsening factors and the number of levels are two closely related parameters. On the one hand, a large

number of levels increases the degree of parallelism and, potentially, the performance of the global search. On the other hand, when coarsening factors increase substantially from one level to the next, the corresponding dimension of the solution space tends to decrease rapidly. Consequently, a high number of levels comes at the cost of low coarsening factor values, which yield small sets of fixing decision variables (blocks). After interacted with adjacent levels, small blocks would conduct low impact on the design. This is the behavior which should be avoided. A compromise between large coarsening factor values and a high number of levels must therefore be established.

This compromise is particularly difficult to find for most CMND problems of interest, where the planned capacity is relatively tight with respect to the total demand (problem instances with almost no slack capacity or, at the other extreme, with almost unlimited capacity are significantly easier to solve). For such problems, increasing the number of fixed-closed variables rapidly yields infeasible instances. Moreover, this characteristic is independent of problem dimensions, which contrasts to other combinatorial problems, such as graph partitioning, where increasing the number of decision variables generally permits to increase the number of levels and the values of the coarsening factors. We decided therefore to fix the number of levels for all problem instances based on the number of design decision variables in the smallest problem instances tested (100 arcs). Experimentally, eight levels appeared as the maximum level of parallelism we could achieve for those instances and this number was used for all calibration and experimentation activities.

To calibrate the coarsening factors, one must consider the fact that fixed-open and fixed-closed arcs do not have the same impact on the exploration of the solution space. During the neighborhood exploration performed by the cycle-based tabu search, cycles that contain fixed-closed arcs are automatically eliminated. On the other hand, a cycle with fixed-open arcs may generate a move if this move does not involve changes to the

PROB	4%	6%	9%
C41(20,300,40,V,L)	429398	430144	430205
C42(20,300,40,F,L)	592267	601831	601636
C43(20,300,40,V,T)	466550	466762	466076
C44(20,300,40,F,T)	608627	615209	615631
C45(20,300,200,V,L)	80619	78173	77653
C46(20,300,200,F,L)	121806	119177	120615
C47(20,300,200,V,T)	79420.7	79045	77906
C48(20,300,200,F,T)	113566	111966	110869

Table 6.1: Calibration of coarsening factors

status of the fixed-open arcs in the cycle. In other words, a fixed-closed status impacts in a more stringent manner the exploration of the solution space than the fixed-open status. Consequently, we considered smaller block sizes for fixed-closed arcs. We tested increasing the total number of fixed arcs per level by 4%, 6%, and 9%, respectively, and coarsening factors with a ratio $1/3$ for the fixed-closed and $2/3$ for the fixed-open blocks, respectively. A coarsening factor of 9% per level, with 3% of the arcs fixed closed and 6% of the arcs fixed open, offered the best results for the calibration tests. Results of testing are displayed in Table 6.1.

6.1.2 Number of tabu iterations between activations of the interaction operators

In Algorithm 5.4 of Chapter 5, we run the modified cycle-based tabu search procedure between each activation of the interaction operators. The purpose is to explore a new solution space obtained by re-coarsening or to re-initialize search in a region of the cur-

rent solution space identified by a solution from an adjacent level. The number of tabu search iterations between activations of the interaction operators impacts on how deeply each run of the tabu search procedure explores one region of the solution space. This also has an impact on the number of times the interaction operators are activated for each run of MCTS. Indeed, the stopping criterion of MCTS is the total number of iterations by the cycle-based tabu search procedure. Therefore, the number of tabu iterations between activations of the interaction operators determines how frequently search at a given level is influenced by adjacent levels. A large value for this parameter induces thorough explorations of specific regions of the solution space. On the other hand, because of the fewer interactions among search threads, a large number of tabu iterations reduces the capacity of the cooperative search to guide explorations to promising regions of the solution space with help from other levels.

For this multilevel cooperative algorithm, we have adopted a simple strategy which consists to run the tabu search procedure for the same fixed number of iterations at each level and for the whole duration of the MCTS computation. We tested the values 5, 10, 20, and 50 for the fixed number of iterations. Results are displayed in Table 6.2, 5 out of 8 problems got better results with value 20. The value 20 has been selected and applied for the whole set of tests in the experimentation phase.

6.1.3 Dimension of the elite solution set

The calibration tests indicated that, in general, differences in the cardinality of the elite solution sets did not impact significantly the solution performance. It also appeared, however, that this cardinality should not be "too high" and that using elite sets that include several solutions dominates elite sets of size one. The rational seems to be the following. When single-elite-solution sets are used, the elite set would be updated sig-

PROB	5	10	20	50
C41(20,300,40,V,L)	429398	429486	429398	430049
C42(20,300,40,F,L)	601006	600271	592267	597597
C43(20,300,40,V,T)	465778	467512	466550	464659
C44(20,300,40,F,T)	619749	615250	608627	608938
C45(20,300,200,V,L)	80657.5	80165.1	80619	80964.3
C46(20,300,200,F,L)	123148	124830	121806	12340
C47(20,300,200,V,T)	79993.5	79656.5	79420.7	80703
C48(20,300,200,F,T)	118536	113486	113566	114010

Table 6.2: Calibration of the number of iterations between interaction operators

nificantly less frequently than when larger sets are used. Consequently, the interpolation and reverse interpolation operators would tend to send the same solution to neighbor levels. On the other hand, large elite sets might contain low-quality solutions which, when transmitted to neighboring levels, would send the exploration at those level astray. A cardinality of 4 for the elite sets appeared a reasonable compromise.

6.1.4 Updating the frequency memory

The goal of the frequency vectors is to accumulate information about the status of decision variables in good solutions. Similarly to the elite set cardinality calibration, a compromise must also be reached in this case. Restricting "good" solutions to those that are very close (less than 5%, say) to the overall best solution enforces the elite characteristic of the frequency vector, but yields procedures where applications of the re-coarsening operator have little impact on coarsening. On the other hand, the update of the frequency vectors based on "poor"-quality solutions implies driving the search

(through the re-coarsening operator) into possibly uninteresting regions of the solution space. A maximum value of 5% less than the cost of the worst solution in the elite set at the corresponding level gave the best performance.

6.1.5 Selecting fixed arcs

We tested the two methods described in Chapter 5 for selecting, during the coarsening phase, arcs which are given a fixed status. The approach based on the frequency memory appeared superior to that based on the *FC* ratio as shown in Table 6.3. In the frequency-memory approach, the initial coarsening takes place once the cycle-based tabu search has failed to improve the best known solution for a number of iterations. We tested values of 20, 50, and 100 for this parameter and a value of 100 performed best as shown in Table 6.4. We impose, however, a limit (fixed at 200) for the maximum number of iterations the search at each level may run in independent mode. Consequently, a level switches to cooperation mode either when it has failed to improve its best solution during the last 100 iterations or when it has reached the 200-iteration limit.

6.2 Experimentation phase

In this experimentation phase, we compare the solutions obtained by our search heuristic with the solutions obtained by the other best known search heuristics for the CMND problem. The two other heuristics are the cycle-based tabu search method [12] and the path relinking algorithm [13]. We run all three heuristics on the same set of benchmark problems and compare the quality of the solutions obtained by each of them.

Our algorithm has been coded in C++. Our tests have been conducted on a Sun Enterprise 10000 with 64 400MHz processors with 64 Gigabyte RAM. The parallel

PROB	FC METHOD	FREQUENCY METHOD
C41(20,300,40,V,L)	432080	429837
C42(20,300,40,F,L)	601973	597137
C43(20,300,40,V,T)	466673	466115
C44(20,300,40,F,T)	618040	610739
C45(20,300,200,V,L)	79566.83	77839.8
C46(20,300,200,F,L)	124438	121242
C47(20,300,200,V,T)	78990	78797.5
C48(20,300,200,F,T)	113430	109590

Table 6.3: Calibration of selection of initial arcs to be fixed

PROB	20	50	100
C41(20,300,40,V,L)	429837	432068	430205
C42(20,300,40,F,L)	601031	59761	601636
C43(20,300,40,V,T)	466061	466580	466076
C44(20,300,40,F,T)	617599	614890	615631
C45(20,300,200,V,L)	78473	78943	77653
C46(20,300,200,F,L)	121463	121700	120615
C47(20,300,200,V,T)	79026	79414	77906
C48(20,300,200,F,T)	114416	112657	110869

Table 6.4: Number of iterations failed to improve the best known solution

programming model for the multilevel procedure is implemented using Pthreads under Solaris 2.7. The exact evaluation of the capacitated multicommodity network flow problem is performed using the linear programming solver CPLEX version 7.5.

The set of benchmark problems are problem instances used by Ghamlouche, Crainic, and Gendreau [12]. They have various sizes up to 700 arcs and 400 commodities. These problem instances are generally transshipment networks with no parallel arcs. Each commodity corresponds to a single origin-destination pair. On each arc, routing costs are the same for all commodities. Problem instances have been generated to provide a variety of ratios of fixed cost to routing cost and ratios of capacity to demand. A detailed description of problem instances is given by Crainic, Frangioni and Gendron [5, 10, 11].

Table 6.5 shows results of the computational experiments. Each problem instance is identified by a number, together with its characterization in terms of the number of nodes, arcs and commodities, as well as representations of fixed cost and capacity type. Compared to the routing cost, a relatively high or low fixed cost is symbolized by the letter **F** or **V**. Compared to the total demand, letters **T** and **L** indicate the problem is tightly or loosely capacitated. Columns **TC** and **AV.TC** report respectively the best and average results out of 3 repetitions of executing the cycle-based tabu search method with 400 iterations [12]. Columns **PR** and **AV.PR** report the best and average results found by the path relinking approach [13] with 3 runs. Columns **MLEVEL** and **AV.MLEVEL** report the best and average results with three times of executions of the MCTS algorithm with 600 iterations (among them, 200 iterations are used for coarsening). All these results are run in the same computing environment.

PROB	TC	PR	AV. TC	AV. PR	MLEVEL	AV. MLEVEL
C1(25,100,10,V,L)	14712	14712	14769.33	14712	14712	14712
C2(25,100,10,F,L)	14941	14941	14941	15081.33	14941	14941
C3(25,100,10,F,T)	50529	49899	50619.67	50154	49937	50287.33
C4(25,100,30,V,T)	365385	365385	365385	365385	365385	365385
C5(25,100,30,F,L)	37515	37654	38672.33	37704	37607	37986.33
C6(25,100,30,F,T)	87325	86428	88095.33	86492.33	86461.3	86643.1
C7(100,400,10,V,L)	28786	28485	28836.67	28529	28486	28513.67
C8(100,400,10,F,L)	24022	24022	24022	24022	24022	24022
C9(100,400,10,F,T)	67184	65278	67376.33	65539.33	65547	66047.67
C10(100,400,30,V,T)	385508	384926	385512.67	385181.67	385280	385289.33
C11(100,400,30,F,L)	51831	51325	52176.33	51875.67	50456	51824.67
C12(100,400,30,F,T)	147193	141359	147478	143403.67	145292	145625.33
C33(20,230,40,V,L)	430628	424385	430768.33	424789.67	426294	426814.67
C35(20,230,40,F,T)	652775	645548	657803.67	646378.67	646227	649620.67
C36(20,230,40,V,T)	372522	371811	372888	372162.33	371475	371754
C37(20,230,200,V,L)	100001	100404	101481.33	101469.33	98582	98727.67
C38(20,230,200,F,L)	148066	147988	148975	151352	143150	144453.33
C39(20,230,200,V,T)	106868	104689	107589.33	105598.67	101860	101938.33
C40(20,230,200,F,T)	147212	147554	147868	148044.33	141188	143173.67
C41(20,300,40,V,L)	432007	429398	432168.33	429523.67	429459	429610.33
C42(20,300,40,F,L)	602180	590427	604072.67	594590.67	593544	595390
C43(20,300,40,V,T)	466115	464509	466546.33	464822.67	464509	465557.67
C44(20,300,40,F,T)	615426	609990	616859.67	609995.33	612162	615014
C45(20,300,200,V,L)	81367	78184	82186.67	79095.33	77495	78093.17
C46(20,300,200,F,L)	122262	123484	123247.67	124924.67	120964	121493
C47(20,300,200,V,T)	80344	78866.80	80724	79212.27	77251	77739.33
C48(20,300,200,F,T)	113947	113584	115342	114632.33	110418	111440.67
C49(30,520,100,V,L)	56603	54904	56722	55043.67	54892.5	55276.83
C50(30,520,100,F,L)	103657	102054	105276.33	104222.33	99817	100374.67
C51(30,520,100,V,T)	54454	53017	54537	53212	53431	53460
C52(30,520,100,F,T)	105130	106130	107885.33	107575	102477	103868.33
C53(30,520,400,V,L)	122673	119416	123277.33	119624	115671	116197.33
C54(30,520,400,F,L)	164140	163112	165458	163377	156363	156450.33
C55(30,520,400,V,T)	122655	120170	123210.33	120764.33	118850	119795.67
C56(30,520,400,F,T)	169508	163675	170301.33	164921.33	160115	160736.33
C57(30,700,100,V,L)	50041	48723	50098.33	48894.33	48750	48809.33
C58(30,700,100,F,L)	64581	63091	64802.67	63436.33	62243	63104.33
C59(30,700,100,V,T)	48176	47209	48249.67	47288.33	47251	47392.67
C60(30,700,100,F,T)	57628	56575.50	58111.33	56808.83	56910	57106
C61(30,700,400,V,L)	107727	105116	108459.33	105403.33	102299	102946.67
C62(30,700,400,F,L)	150256	145026	150909	147887.33	141254	143065
C63(30,700,400,V,T)	101749	101212	103112.33	102119.33	99194.9	99481.77
C64(30,700,400,F,T)	144852	141013	146705	141446.67	136350	137660.67

Table 6.5: Computational results

6.2.1 Result analysis

Experimental results indicate that the MCTS algorithm outperforms the cycle-based tabu search method for almost all problem instances. Observing the columns **MLEVEL** and **TC**, out of 43 problem instances, the MCTS algorithm displays a worse solution for only one problem instance. For all the other problem instances, the MCTS algorithm provide solutions better than or equal to ones using the cycle-based tabu search method. In columns **AV.MLEVEL** and **AV.TC** reporting the average best solution out of 3 runs, for all the problem instances, the MCTS algorithm obtained solutions superior to the solutions obtained by the cycle-based tabu search algorithm.

Compared to the path relinking algorithm, the best current meta-heuristic for the CMND, the MCTS method displays very competitive results. Results in columns **MLEVEL** and **PR** show that out of 43 problem instances, solutions of 28 problems identified by the MCTS method are better than or equal to those obtained by path relinking. Columns **AV.MLEVEL** and **AV.PR** illustrate that improved solutions of 27 problems identified by the multilevel method are found compared to solutions identified by path relinking.

Among problem instances with a large number of commodities (200 and 400), MCTS is even more impressive. Results have proved that, among 16 problem instances with commodities 200 and 400, all solutions identified by the MCTS method are better than those identifed by both the cycle-based tabu search algorithm and path relinking no matter for the best solution comparasion or the average best solution comparison. Problems with a large number of commodities are usually more difficult to solve due to the competition of limited capacities for all commodities. The fact that MCTS performs better for these problem instances is quite significant. Search heuristics tend to reach a ceiling in terms of the quality of the approximations they can get or the size of the problems they can solved efficiently. Beyond this ceiling, increasing substantially the computational time

have only a marginal impact in terms of improving the quality of the solution. One has to wait for a new heuristic method to go beyond that ceiling. MCTS could be the new heuristic method that will allow to solve more difficult CMND problems.

The multilevel method is less impressive on problem instances C1 to C12. These performances are caused by the large number of infeasible solutions experienced by the multilevel procedures for this sequence of problem instances. We observe the followings: the total number of solutions visited by the 8 levels is $8 \times 600 = 4800$. The ratio of infeasible solutions for the multilevel search is C2 = 62%, C3 = 68%, C6 = 34%, C8 = 81%, C9 = 70% and C12 = 45%. In comparison, the ratio of infeasible solutions is C3 = 7% and C12 = 22% for the parallel independent search method. Clearly, the ratio of infeasible solutions is much higher for the multilevel method than parallel independent search, consequently much higher than for the sequential cycle-based tabu search method. For problem instances C2, C3, C8 and C9, the number of infeasible solutions is almost the same at all levels while there are more infeasible solutions at higher levels for problem instances C6 and C12. For all the above problem instances, the density of infeasible solutions for the multilevel method increases with the number of iterations. By comparing the ratios above, one can also observe that when all problem parameters are identical except for the number of commodities (C3, C6), an increase in the number of commodities reduces the number of infeasible solutions (C3 = 68% and C6 = 34%). When the number of commodities is the same (C2, C3, C8 and C9 have 10 commodities), the ratio of infeasible solutions is larger when the number of arcs is larger (C2, C3 have 100 arcs and C8 and C9 have 400 arcs). This behavior has also been observed for problems with 30 commodities (C6 and C12), C6 with 100 has fewer infeasible solutions than C12 with 400 arcs.

For problem instances larger than C1 to C12, the situation regarding infeasible solutions substantially improves. For example, C33 = 8%, C35 = 11%, C36 = 24%, C41 =

10%, C43 = 21%, C44 = 20%. But this ratio is still higher then the number of infeasible solutions for independent search. For independent search and problem instance C44, we have approximated this ratio by multiplying by 8 the number of infeasible solutions of the sequential cycle-based tabu search on 600 iterations. The ratio for C44 is 3%. For these problem instances, there seems to be some dividing line between problems instances with tight capacity (C35, C36, C43 and C44) and those loose capacity on the links. The number of infeasible solutions is greater when the capacities are tight. For all these problem instances, the number of infeasible solutions is larger at levels 0 and 1. Finally, for really large problems, such 520 or 700 arcs instances, and 100 or 400 commodities, the number of infeasible solutions tends to be the same as for parallel independent search. This ratio is smaller than 1% for both C56 and C61 in multilevel and has been approximated to be about the same ratio for parallel independent search.

The interpretation of these observations goes along the following lines. There are some correlations between the size of the problem instance and the number of infeasible solutions even for the sequential tabu search method. For smaller problem instances such as C1 to C12, we suspect that calibration, in particular calibration of the number of levels and the coarsening factors, almost certainly worsen the situation regarding infeasible solutions. No doubt that the stringent coarsening factors of our calibration as well as the number of levels play also a major factor for the occurrence of infeasible solutions. Fixing arcs, particularly fixed-closed arcs, reduces the search space and hence increases the likelihood of moves including artificial arcs. Hence, when high coarsening factors are used, e.g., for higher levels, the search space becomes very constrained and many moves produce infeasible solutions.

6.2.2 Search behavior analysis

We have designed MCTS by adapting the coarsening and refinement phases of multilevel cooperative search to the CMND problem. This MCTS algorithm is mainly a proof that multilevel cooperative search paradigm can be a successful approach for the CMND problem. The experimental results show this is indeed a promising approach in comparison with other heuristics for the CMND problem. To improve the design of MCTS and to make it more efficient, we proceed to analyze the search behavior of MCTS. This analysis will help to illustrate the algorithm described in Chapter 5. It will also initiate the fine tuning work of MCTS by showing how coarsening and interaction operators can be used more efficiently or better adapted to the CMND problem.

We illustrate this analysis by charting the evolution of the parallel search for 3 representative problem instances. Figures 6.3, 6.4 and 6.5 display the evolution of the best solution value at each of the 8 levels of problem instance C45, C56 and C61 for 600 iterations of the tabu search method. For comparison purposes, Figure 6.2 displays the same evolution results for a parallel independent algorithm running 8 cycle-based tabu search procedures for 600 iterations. In all these figures, solution costs reaching the upper limit of charts indicate infeasible solutions. Moreover, figures don't hold the order of time for all levels, i.e., the solution of iteration 89 for level 6 may be found before the solution of iteration 20 for level 0.

At early stages of the search, for each level, the MCTS procedure executes the cycle-based tabu search independently. During that phase, each independent cycle-based tabu search builds the containts of its frequency memory and elite set. This phase of independent search ends after a pre-determined number of cycle-based tabu search without improvement or once the cycle-based tabu search has executed 200 iterations. Then the containts of the frequency memories is used by the frequency method to coarsen the

original problem instance. Once a process i ends its independent search phase, it performs coarsening and starts to run in cooperative mode. It does not matter in this case whether processes $i+1$ and $i-1$ are still running in independent mode. Processes $i+1$ and $i-1$ have frequency memories and elite sets ready, therefore process i can activate its interaction operators without any problem.

This phase of independent search appears distinctively in the 3 charts of cooperative search. In each of those charts, the cost function has an interval of improvement followed by an interval of increase in the cost of the solutions visited. It is also during that phase that we see a more distinct separation between processes performing tabu search intensification (essentially level 0 and 1) and the other processes without intensification. Levels 0 and 1 display steeper descents than the other levels. Without the benefit of intensification, the other levels fail to identify the good solutions in the regions of the solution space they visit. Once cooperation kicks in, components of the solutions discovered by intensification at levels 0 and 1 are transfered to coarsening at the other levels, which helps those levels to navigate in better regions of the solution space.

The multilevel method exhibits a marked drop in solution value between iterations 150 to 250, that is, when the initial blocks of fixed decision variables are generated and inter-level cooperations starts to kick in. This drop may be noticed at all levels, but it is more accentuated at levels 0, 1, and 2 where the search has more degrees of freedom (smaller blocks). The marked improvement brought by the initiation of cooperation is followed by a more gradual improvement in the best-solution value. This behavior may be observed for all problem instances and is a clear indication of the impact of the cooperation mechanism. To illustrate this behavior, we follow the evolution of the search in Figure 6.4.

For this problem instance, level 0 finds a very good initial solution at iteration 46, while running in independent mode. Recall that at level 0 one does not generate blocks

of fixed decision variables, nor perform reverse interpolation or re-coarsening local interactions. Consequently, the drop that occurs at iteration 147 is due solely to the interpolation operator that imports the best elite solution from level 1. The best initial solution for level 1 is found at iteration 52 and the interactions start at iteration 152: fixed-open and fixed-close blocks defined, followed by an execution of the interpolation operator with level 2. The improvement that occurs at iterations 154 and 155 may be caused either by the tabu search intensification phases performed on the imported solution from level 2 or by the restart from the best elite solution of level 2 (which, by observing the chart, is not very good). The similar behavior observed at levels 2 to 6 (roughly between iterations 153 and 185) can only be caused by the interpolation operator and the generation of blocks of fixed decision variables. As for level 7, the highest one, it does not perform any interaction after the initial generation of the set of blocks (iteration 181) and the improvement observed at iteration 184 is caused strictly by the large blocks of fixed decision variables.

A significant difference may be observed between the independent and the multilevel parallel searches, as illustrated in Figures 6.2 to 6.5. Any thread participating in a parallel independent search may identify good solutions, based on its degree of success in the random selection of neighbors by the cycle-based tabu search method. This is expected as the search threads are independent from one another. The situation is different for the multilevel method, where the lowest levels are almost exclusively the ones where good solutions are found. Thus, in our examples, most improving solutions are found at levels 0 to 2, while levels 3 and 4 sometimes obtain solutions close to the best ones, levels 6 and 7 contributing very little due to the large number of fixed arcs (as shown in Figure 6.1). This characteristic of the multilevel search follows from the combined effect of local interactions, the hierarchical structure of neighborhoods, and the tabu search intensification phases used at levels 0 and 1. This observation does not

mean that higher levels are not useful. On the contrary, they contribute to diversify the search.

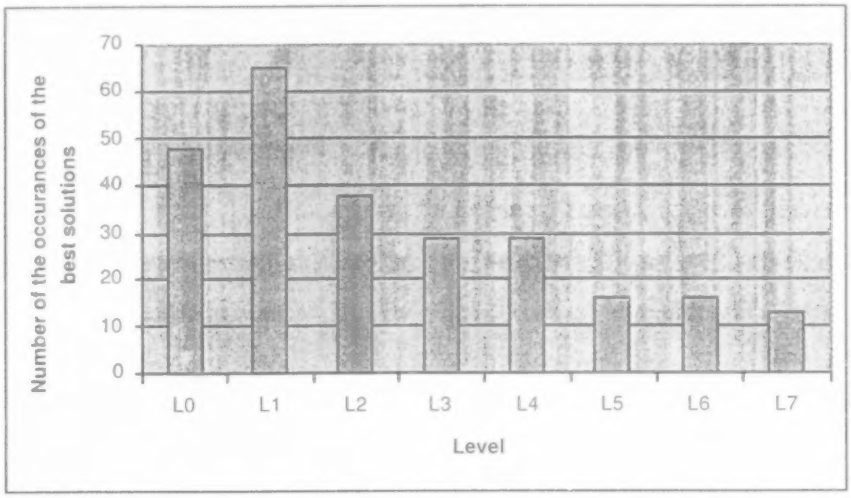


Figure 6.1: Number of occurrences of the best solutions at each level

A second difference between independent and multilevel searches is apparent in the figures: Functions cross less often each other in multilevel search than for independent search. One may infer than that the multilevel neighborhood structure based on increasing sizes of blocks is relatively successful at preventing overlaps among parallel explorations of the solution space. Moreover, the gradual improvement observed once the cooperation is active supports the claim that cooperation by local interactions among levels is successful in driving levels 0 and 1 towards good regions of the solution space.

In some problem instances, the independent search phase of multilevel corresponds to an ascent rather than a descent as for the problems studied in the previous section. This is the case for example for C33 and C49. For these two problem instances, the kick down phase is observed much sooner, around iterations 100 to 120. It is also during this

kick down phase where the best solutions the multilevel method are observed. Beyond this phase the multilevel method is very erratic, lower levels trailing sometime for more than 60 consecutive iterations in poor regions of the solution space while, on the other hand, higher levels make occasional brief picks in regions of the solution space as good as those visited by the lowest levels. We have no way at this point to make sense of this behaviour, both for the ascent of the cycle-based search and the multilevel procedure.

The key element impacting negatively the adaptation of the multilevel method to the CMND problem is most likely the re-coarsening operator. One of the major reasons for adding coarsening and re-coarsening operators to cooperative search based on local interactions is to increase the stability and smoothness of the search. Cooperative procedures based only on restarts tend to be unreliable, doing well on some problem instances and poorly on others, even failing to beat a sequential search. When comparing the computational results of the multilevel method for the CMND problem and those for the graph and hypergraph problems, we observe a major difference in the degree of stability and smoothness of the two types of adaptations. For graph and hypergraph problems, the multilevel method reaches an equilibrium point where all the levels have about the same quality of solution. When that happens, local interactions no longer trigger changes in the neighborhood structures and restarts of the search at each level. At that point, the multilevel procedure has satisfied a natural stopping criterion. To reach such equilibrium points, the levels should smoothly improve the quality of solutions discovered. In Figure 6.5, where levels appear more distinctly, we observe that the explorations in levels 5, 6 and 7 retreats in worst regions of the solution space. This is the opposite of having a smooth convergence. The levels are not playing their role with respect to the multilevel method. This is certainly a major factor for having solutions worst than the sequential cycle-based tabu. The failure of the re-coarsening operator is probably related to a complex interplay between the CMND problem structure, the

cycle-based tabu search method and factors related to the design strategy of the multilevel method. For example, we need to evaluate if the tabu intensification phase at levels 0 and 1 defeats to some extend the purpose of re-coarsening, and if so, how we can address this problem in the design of re-coarsening operators. We need to study these problems in order to get more efficient re-coarsening operators, this is certainly one important task for the future development of the multilevel method to the CMND problem.

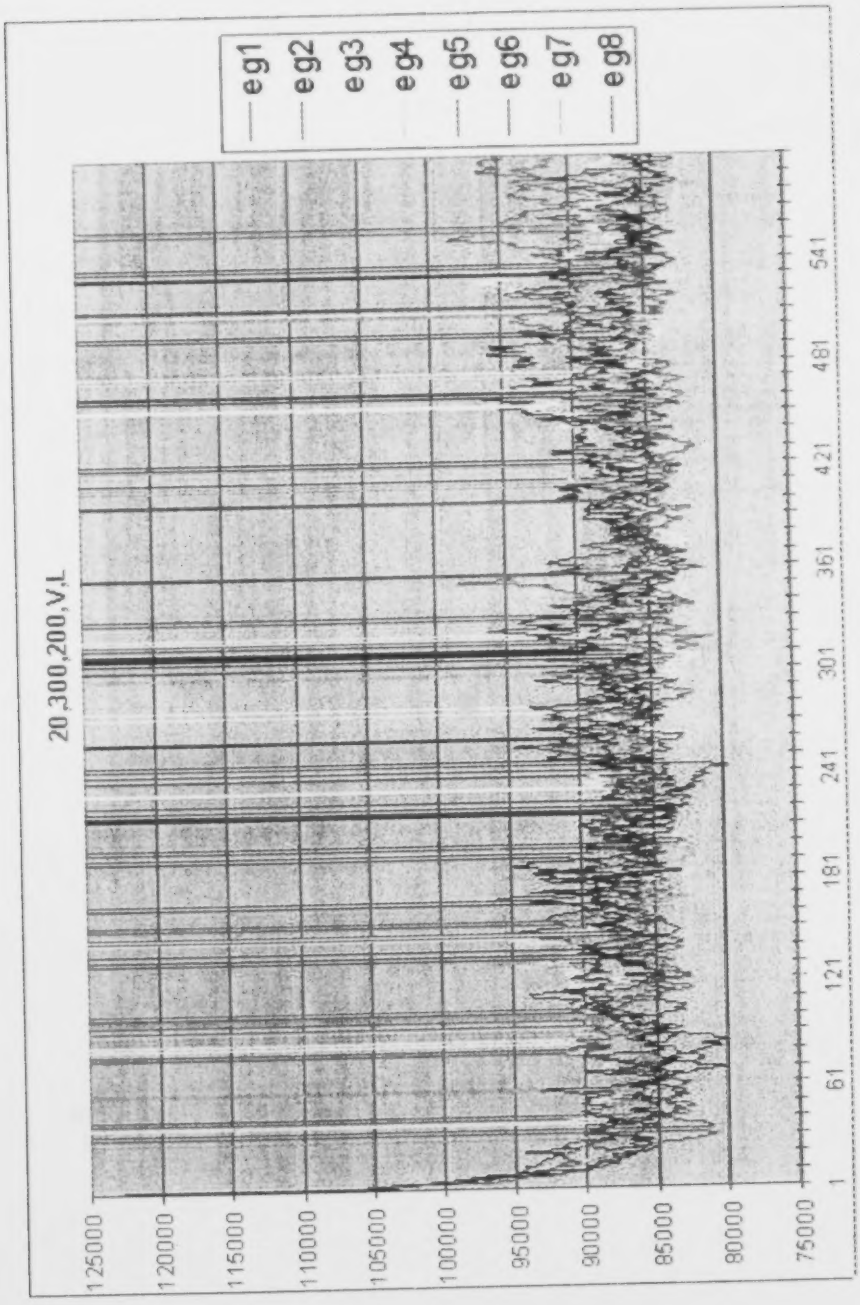


Figure 6.2: Chart of solutions for parallel independent search for problem C45

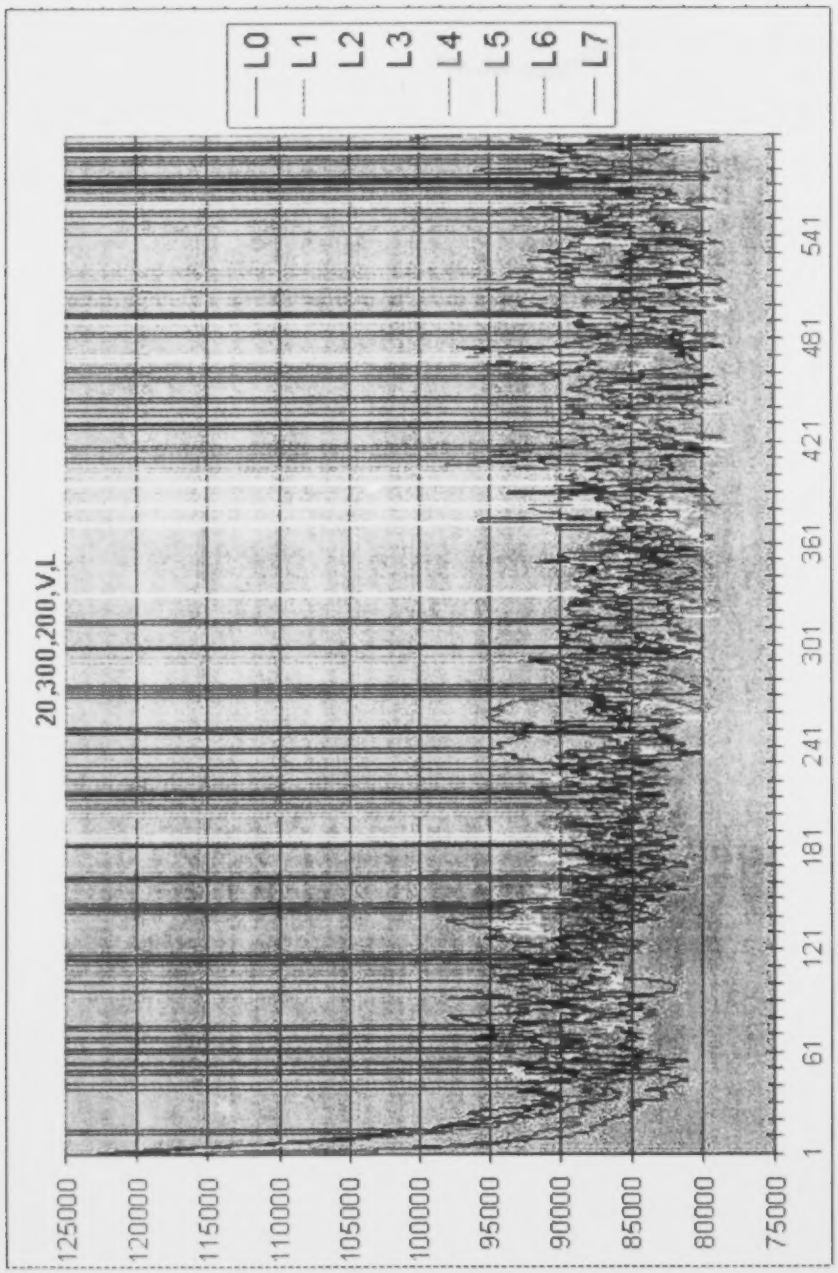


Figure 6.3: Chart of solutions for MCTS and problem C45

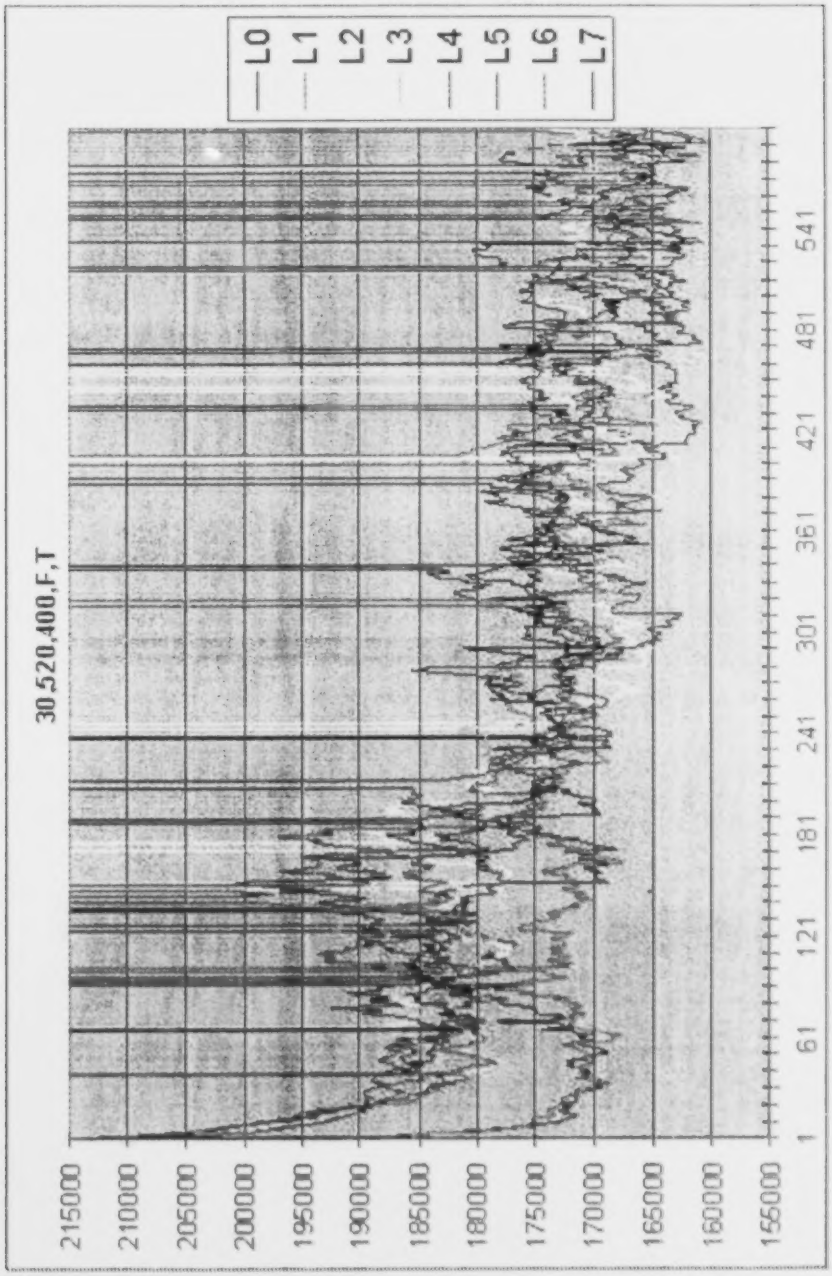


Figure 6.4: Chart of solutions for MCTS and problem C56

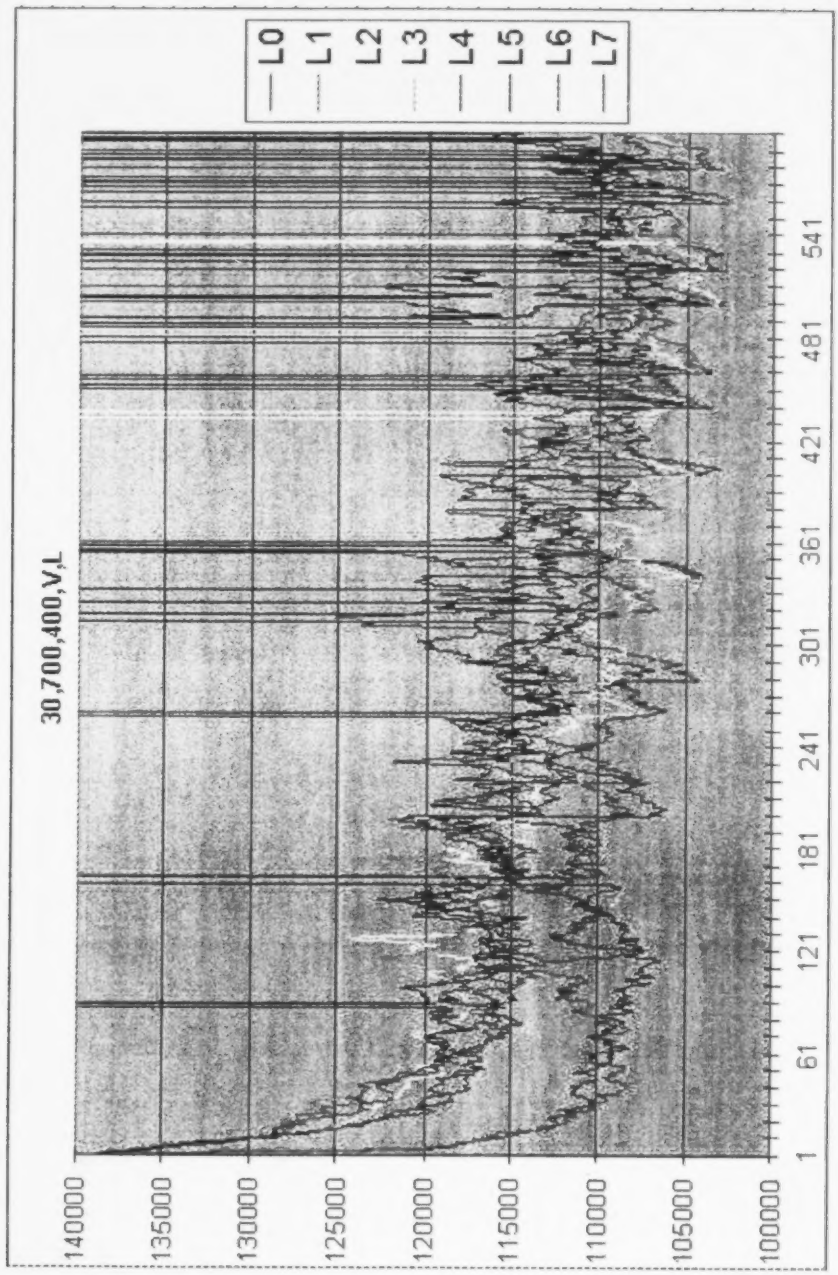


Figure 6.5: Chart of solutions for MCTS and problem C61

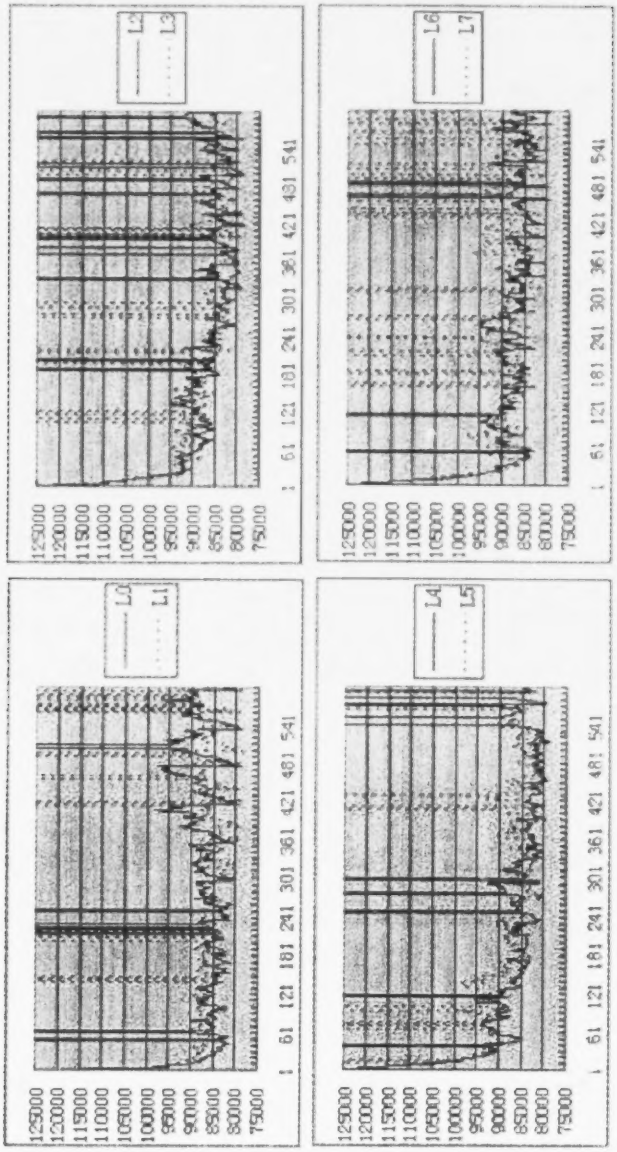


Figure 6.6: Chart of solutions for MCTS and problem C45

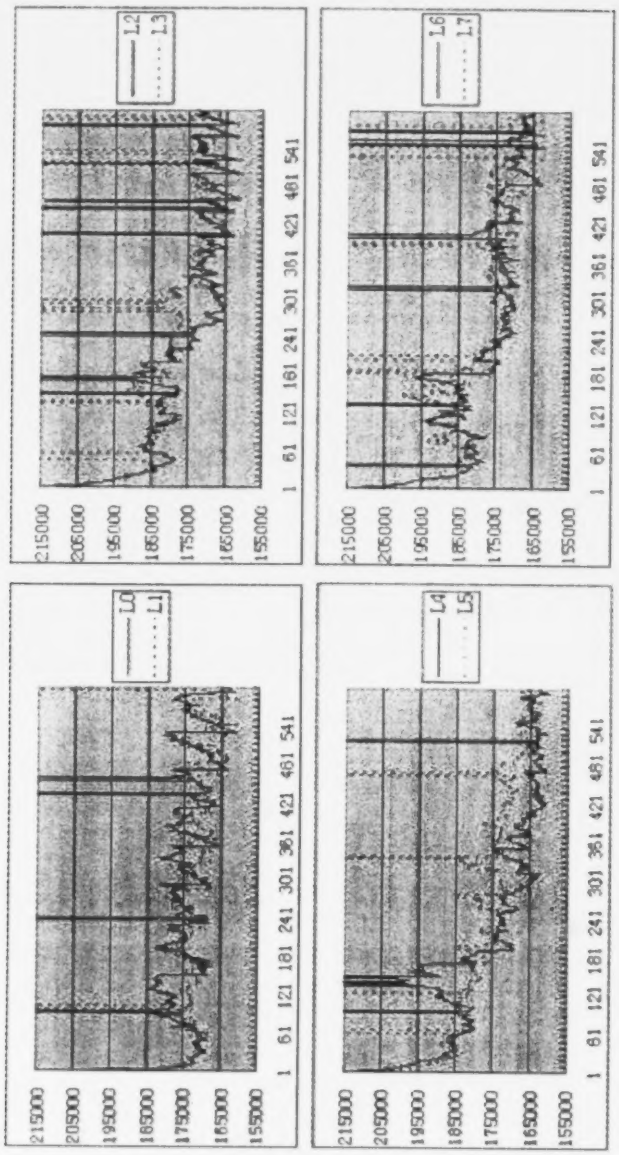


Figure 6.7: Chart of solutions for MCTS and problem C56

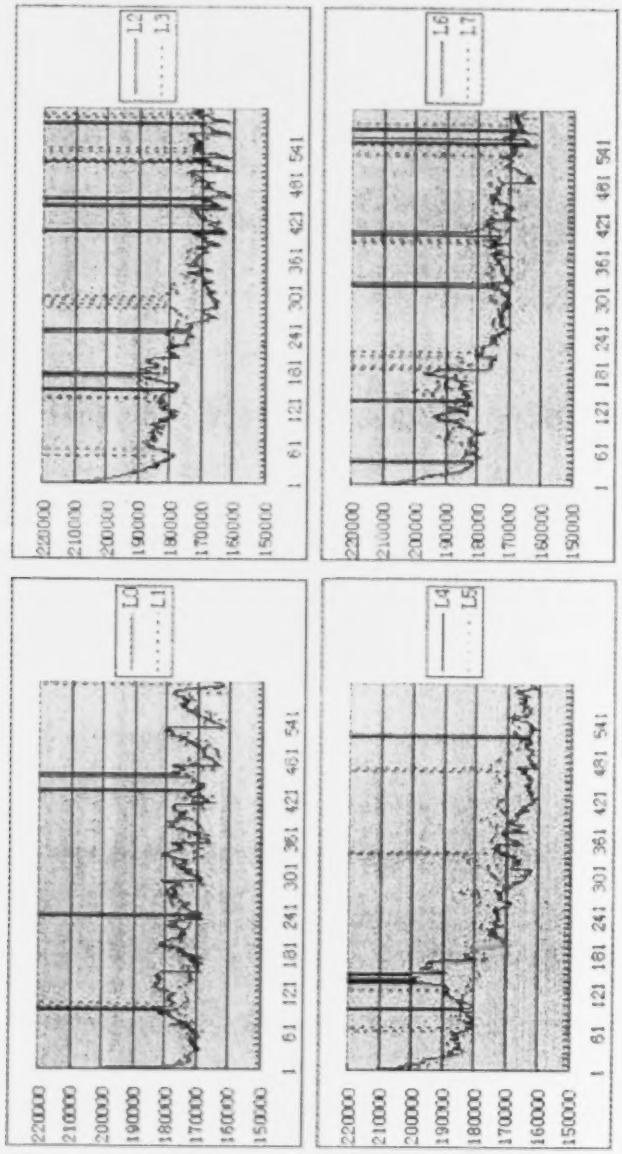


Figure 6.8: Chart of solutions for MCTS and problem C61

Chapter 7

Conclusions and Future Work

We presented the first multilevel cooperative parallel tabu search algorithm for the capacitated multicommodity network design problem. Computational results on a set of benchmark problem instances show that this approach yields good quality solutions comparable to those obtained by the current best meta-heuristics for the problem.

These results are very encouraging but they also emphasize the need for more research to understand better the interplay between the CMND problem, the cycle-based tabu search, and the multilevel strategy for cooperative parallel search, and to develop the next generation of algorithms.

The development of this first multilevel cooperative parallel meta-heuristic for the CMND problem, together with the results of the numerical experiments that were performed, pointed to a number of research issues that should conduct to an enhanced methodology.

A number of issues have been highlighted by the experiments. For example, a relatively high number of infeasible solutions were generated. Also, searches proceeding at high levels (e.g., levels, 5, 6, and 7) were not very productive in terms of generating

good solutions. We also noticed that the searches at the various levels did not always proceed at the same pace, and produced networks with dissimilar designs. We believe that one of the main reasons behind this behavior is the very (probably, too) simple and stringent definition of the cooperation mechanism, particularly the coarsening operation and factors.

Indeed, the cycle-based neighborhood definition is prone to yield moves that bring the search to infeasible solutions (see the example in Ghamlouche, Crainic, and Gendreau [12]). Fixing arcs, particularly fixing them closed, reduces the search space and increases the likelihood of such moves, especially for the problem instances tested, characterized by a tight total capacity compared to total demand. Then, when large coarsening factors are used for high levels, the search space becomes too constrained and many moves conduct to infeasible solutions. On the other hand, fixing arcs modifies the problem instance and thus the behavior of the tabu search. Thus, when computing cycles, the cost of moving flow on a fixed-open arc is equal to the variable cost, while this cost is weighted by a linearization of the fixed cost when the arc is undecided. This may bias the search towards using the fixed arcs and not allow a thorough exploration of the neighborhood.

One must also recall that, for CMND problems, the search proceeds in the space of the design variables only. The feasibility of the design resulting from a move as well as the associated value of the design and flow distribution must therefore be computed by solving a rather complex capacitated, multicommodity minimum-cost network flow problem. The impact of aggregating variables in blocks to define levels is therefore much more difficult to determine than, for example, in the case of the graph partitioning problem. Block building may thus set the search in a not-so-interesting region of the solution space of the original problem and many interactions may be required before the search moves toward a more interesting region.

A challenging research direction corresponds therefore to the study of coarsening operators. The ones presented in this thesis are quite simple, as they are based on characteristics of individual arcs. More elaborate coarsening operators ones could exploit the characteristics of the multicommodity network design problem, e.g., the paths or trees used to move commodities. The relations between the problem characteristics, in terms of dimensions and capacity, on the one hand, and the number of levels and coarsening operators and factors, on the other hand, should also be part of this study.

A second, and complementary, research direction concerns the flexibility of the cooperation structure. Currently, the mechanism is quite strict. The operators are invoked at regular intervals, in strict order. The literature on parallel meta-heuristics indicates, however, that introducing flexibility in the algorithms is beneficial. Thus, while the strict mechanism used in the present thesis yields very good results, a study of more advanced mechanisms that would allow the exploration to dynamically adapt to the problem instance and the state of the search is needed.

Exploring in this direction brings up issues related to the utilization of memories in cooperation. Simple memories are already included in the algorithm and used by the interaction operators. More advanced memories may be built. Indeed, similar to solution values, other solution and exploration attributes may be put into memories and associated to interaction operators. Such data may even be part of the information diffusion process. The utilization of this diffusion to create or approximate at each level an image of the status of the entire search is a fascinating challenge.

Bibliography

[1] A. Balakrishnan, T.L. Magnanti, and P. Mirchandani. Network design. In M. Dell'Amico, F. Maffioli, and S. Martello, editors, *Annotated Bibliographies in Combinatorial Optimization*, pages 311–334. John Wiley Sons, New York, NY, 1997.

[2] S. T. Barnard and H. D. Simon. A fast multilevel implementation of recursive spectral bisection for partitioning unstructured problems. *Concurrency: Practice & Experience*, 6(2):101–117, 1994.

[3] D. Bienstock and O. Günlük. Capacitated network design-polyhedral structure and computation. *IMFORMS Journal on Computing*, 8(3):243–259, 1996.

[4] S. H. Clearwater, B. A. Huberman, and T. Hogg. Cooperative solution of constraint satisfaction problems. *Science*, 254:1181–1183, 1991.

[5] T.G. Crainic, A. Frangioni, and B. Gendron. Bundle-based relaxation methods for multicommodity capacitated network design. *Discrete Applied Mathematics*, 112:73–99, 2001.

[6] T.G. Crainic and M. Gendreau. Cooperative parallel tabu search for capacitated network design. *Journal of Heuristics*, 8(6):601–627, November 2002.

[7] T.G. Crainic, M. Gendreau, and J.M. Farvolden. A simplex-based tabu search method for capacitated network design. *INFORMS Journal on Computing*, 12(3):223–236, 2000.

[8] T.G. Crainic, M. Toulouse, and M. Gendreau. Towards a taxonomy of parallel tabu search heuristics. *INFORMS Journal on Computing*, 9(1):61–72, 1997.

[9] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, New York, 1979.

[10] B. Gendron and T.G. Crainic. Relaxations for multicommodity network design problems. Publication CRT-965, Centre de recherche sur les transports, Université de Montréal, Montréal, QC, Canada, 1994.

[11] B. Gendron and T.G. Crainic. Bounding procedures for multicommodity capacitated network design problems. Publication CRT-96-06, Centre de recherche sur les transports, Université de Montréal, Montréal, QC, Canada, 1996.

[12] I. Ghamlouche, T.G. Crainic, and M. Gendreau. Cycle-based neighbourhoods for fixed-charge capacitated multicommodity network design. *Operations Research*, 51(4):655–667, 2003.

[13] I. Ghamlouche, T.G. Crainic, and M. Gendreau. Path relinking, cycle-based neighbourhoods and capacitated multicommodity network design. *Annals of Operations Research*, 131:109–133, 2004.

[14] B. Hendrickson and R. Leland. A multilevel algorithm for partitioning graphs. Report SAND93-1301, Sandia National Laboratories, 1993.

[15] T. Hogg and C. Williams. Solving the really hard problems with cooperative search. In *Proceedings of the 11th National Conference on Artificial Intelligence (AAAI93)*, pages 231–236. AAAI Press, Aug. 1993.

[16] B. A. Huberman. The performance of cooperative processes. *Physica D*, 42:38–47, 1990.

[17] D. S. Johnson, C. R. Aragon, L. A. McGeoch, and C. Schevon. Optimization by simulated annealing: an experimental evaluation. part i, graph partitioning. *Oper. Res.*, 37(6):865–892, 1989.

[18] G. Karypis, R. Aggarwal, V. Kumar, and S. Shekhar. Multilevel hypergraph partitioning: Application in VLSI domain. *IEEE Transactions on VLSI Systems*, 7(1):69–79, March 1999.

[19] G. Karypis and V. Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM J. Sci. Comput.*, 20(1):359–392, 1998.

[20] B.W. Kernighan and S. Lin. An efficient heuristic porcedure for partitioning graphs. *The Bell System Technical J.*, 49:291–307, 1970.

[21] Y.A. Koskosidis, W.B. owell, and M.M. Solomon. An optimization-based heuristic for vehicle routing and scheduling with soft time windows constraints. *Transportation Science*, 26:69–85, 1992.

[22] K-G. Lee and S-Y. Lee. Efficient parallelization of simulated annealing using multiple markov chains: An application to graph partitioning. In *Proc. 1992 of the Int. Conf. on Parallel Processing*, pages III 177–180, Aug. 1992.

[23] D. Levine. A parallel genetic algorithm for the set partitioning problem. In *Meta-Heuristics: Theory and Applications*, pages 23–35. Kluwer Academic Publishers, 1996.

[24] T.L. Magnanti and R.T. Wong. Network design and transportation planning: Models and algorithms. *Transportation Science*, 18(1):1–55, 1984.

[25] M. Minoux. Network synthesis and optimum network design problems: Models, solution mehods and applications. *Networks*, 19:313–360, 1989.

[26] M. Ouyang, M. Toulouse, K. Thulasiraman, F. Glover, and J.S. Deogun. Multilevel cooperative search for the circuit/hypergraph partitioning problem. *IEEE Transactions on Computer-Aided Design*, 21(6):685–693, 2002.

[27] W.B. Powell. A local improvement heuristic for the design of less-than-truckload motor carrier networks. *Transportation Science*, 20(4):246–357, 1986.

[28] E. Rolland, H. Pirkul, and F. Glover. Tabu search for graph partitioning. *Ann. Opr. Res.*, 63:209–232, 1996.

[29] V. Schnecke and O. Vornberger. An adaptive parallel genetic algorithm for VLSI-layout optimization. In *Proceedings of the Fourth Workshop on Parallel Problem Solving from Nature*, pages 859–868. Springer - Verlag, 1996.

[30] E. Taillard. Parallel taboo search techniques for the job shop scheduling problem. *ORSA Journal on Computing*, 6(2):108–117, 1994.

[31] L. Tao, Y.C. Zhao, K. Thulasiraman, and M.N.S. Swamy. Simulated annealing and tabu search algorithms for multiway graph partition. *Jounal of Circuits, Systems, and Computers*, 2(2):159–185, 1992.

[32] M. Toulouse, T. Crainic, and K. Thulasiraman. Global optimization properties of parallel cooperative search algorithms: A simulation study. *Parallel Computing*, 26:91–112, 2000.

[33] M. Toulouse, T.G. Crainic, and B. Sansó. Self-organization in cooperative tabu search algorithms. In *1998 IEEE International Conference on Systems, Man and Cybernetics*, pages 2379–2385. Omnipress, Oct. 1998.

[34] M. Toulouse, T.G. Crainic, and B. Sansó. An experimental study of systemic behavior of cooperative search algorithms. In S. voss, S. Martello, I.H. Osman, and C. Roucairol, editors, *Meta-Heuristics: Advances and Trends in Local Search Paradigms for Optimization*, pages 373 -392. Kluwer Academic Publishers, 1999.

[35] M. Toulouse, T.G. Crainic, and B. Sansó. Systemic behavior of cooperative search algorithms. *Parallel Computing*, 30(1):57–79, 2004.

[36] M. Toulouse, K. Thulasiraman, and F. Glover. Multi-level cooperative search: A new paradigm for combinatorial optimization and an application to graph partitioning. In *5th International euro-par parallel processing conference, volumme 1685 of Lecture notes in Computer Science*, pages 533–542, 1999.

[37] C. Walshaw. A multilevel approach to the graph colouring problem. Tech. Rep. 01/IM/69, Comp. Math. Sci., Univ. Greenwich, London SE10 9LS, UK, May 2001.

[38] C. Walshaw. A multilevel approach to the travelling salesman problem. *Oper. Res.*, 50(5):862–877, 2002. (originally published as Univ. Greenwich Tech. Rep. 00/IM/63).